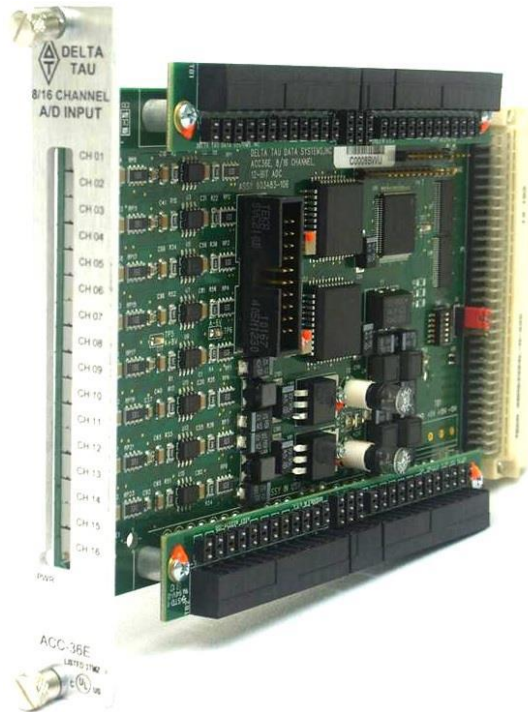


# USER MANUAL

## Accessory 36E



Analog I/O – 16 x 12-bit Inputs

3AX-603483-XUXX

February 24, 2014



**DELTA TAU**  
Data Systems, Inc.

*NEW IDEAS IN MOTION ...*

## Copyright Information

© 2014 Delta Tau Data Systems, Inc. All rights reserved.

This document is furnished for the customers of Delta Tau Data Systems, Inc. Other uses are unauthorized without written permission of Delta Tau Data Systems, Inc. Information contained in this manual may be updated from time-to-time due to product improvements, etc., and may not conform in every respect to former issues.

To report errors or inconsistencies, call or email:

**Delta Tau Data Systems, Inc. Technical Support**

Phone: (818) 717-5656

Fax: (818) 998-7807

Email: [support@deltatau.com](mailto:support@deltatau.com)

Website: <http://www.deltatau.com>

## Operating Conditions

All Delta Tau Data Systems, Inc. motion controller products, accessories, and amplifiers contain static sensitive components that can be damaged by incorrect handling. When installing or handling Delta Tau Data Systems, Inc. products, avoid contact with highly insulated materials. Only qualified personnel should be allowed to handle this equipment.

In the case of industrial applications, we expect our products to be protected from hazardous or conductive materials and/or environments that could cause harm to the controller by damaging components or causing electrical shorts. When our products are used in an industrial environment, install them into an industrial electrical cabinet or industrial PC to protect them from excessive or corrosive moisture, abnormal ambient temperatures, and conductive materials. If Delta Tau Data Systems, Inc. products are directly exposed to hazardous or conductive materials and/or environments, we cannot guarantee their operation.



A Warning identifies hazards that could result in personal injury or death. It precedes the discussion of interest.

**WARNING**

---



A Caution identifies hazards that could result in equipment damage. It precedes the discussion of interest.

**Caution**

---



A Note identifies information critical to the user's understanding or use of the equipment. It follows the discussion of interest.

**Note**

---

REVISION HISTORY				
REV.	DESCRIPTION	DATE	CHG	APPVD
1	ADDED CE DECLARATION	06/07/06	CP	SF
2	ADDED SAMPLE WIRING DIAGRAM	09/08/06	CP	SS
3	ADDED UL SEAL TO MANUAL COVER UPDATED AGENCY APPROVAL/SAFETY SECTION	09/30/09	CP	SF
5	MANUAL REFORMATTING; ADDED POWER PMAC SECTION	10/15/10	DCDP	RN
6	REVISED UMAC MACRO SECTION	4/21/11	DCDP	RN
7	CORRECTED ADCDEMUX INSTRUCTIONS	2/24/14	DCDP	RN

This page left blank intentionally

## Table of Contents

<b>INTRODUCTION.....</b>	<b>7</b>
<b>SPECIFICATIONS.....</b>	<b>8</b>
Environmental Specifications .....	8
Electrical Specifications .....	8
Physical Specifications.....	9
<b>ADDRESSING ACC-36E .....</b>	<b>10</b>
Turbo/Power UMAC, MACRO Station Dip Switch Settings .....	10
Legacy MACRO Dip Switch Settings .....	10
Hardware Address Limitations .....	11
<b>USING ACC-36E WITH TURBO UMAC.....</b>	<b>12</b>
Setting Up the Analog Inputs (ADCs) .....	12
<i>Automatic ADC Read.....</i>	<i>13</i>
<i>Manual ADC Read .....</i>	<i>16</i>
<i>Testing the Analog Inputs .....</i>	<i>20</i>
<i>Using an Analog Input for Servo Feedback.....</i>	<i>21</i>
<i>Analog Input Power-On Position.....</i>	<i>22</i>
<b>USING ACC-36E WITH POWER UMAC .....</b>	<b>23</b>
Setting Up the Analog Inputs (ADCs) .....	23
<i>Automatic ADC Read.....</i>	<i>24</i>
<i>Using an Analog Input for Servo Feedback.....</i>	<i>29</i>
<i>Analog Input Power-On Position.....</i>	<i>30</i>
<i>Manual ADC Read Using ACC-36E Structures.....</i>	<i>31</i>
<i>Accessing ADCs from C Environment (For C Programmers).....</i>	<i>34</i>
<i>Testing the Analog Inputs .....</i>	<i>40</i>
<b>USING ACC-36E WITH UMAC MACRO.....</b>	<b>41</b>
Quick Review: Nodes and Addressing.....	42
Enabling MACRO Station ADC Processing .....	44
<i>Transferring Data over I/O Nodes .....</i>	<i>47</i>
<i>Automatic I/O Node Data Transfer: MI173, MI174, MI175 .....</i>	<i>48</i>
<i>Manual I/O Node Data Transfer: MI19...MI68.....</i>	<i>54</i>
<i>Using an Analog Input for Servo Feedback over MACRO.....</i>	<i>61</i>
<i>Analog Input Power-On Position over MACRO.....</i>	<i>62</i>
<b>ACC-36E LAYOUT &amp; PINOUTS.....</b>	<b>63</b>
Sample Wiring Diagram.....	65
P1: Backplane Bus .....	66
P3 .....	66

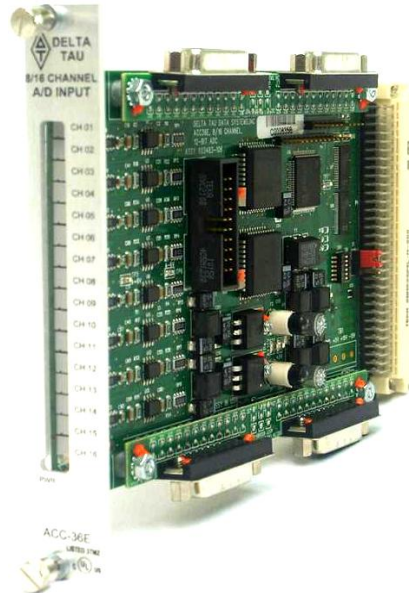
TB1 (4-Pin Terminal Block).....	66
DB15 Breakout Option.....	67
<i>J1/J2 Top, J1/J2 Bottom</i> .....	67
<i>J1 Top: ADC1 through ADC4</i> .....	68
<i>J2 Top: ADC5 through ADC8</i> .....	68
<i>J1 Bottom: ADC9 through ADC12</i> .....	69
<i>J2 Bottom: ADC13 through ADC16</i> .....	69
Terminal Block Option.....	70
<i>TB1 Top: ADC1 through ADC4</i> .....	70
<i>TB2 Top: ADC5 through ADC8</i> .....	70
<i>TB3 Top: Power Supply Outputs</i> .....	71
<i>TB1 Bottom: ADC9 through ADC12</i> .....	72
<i>TB2 Bottom: ADC13 through ADC16</i> .....	72
<i>TB3 Bottom: Power Supply Outputs</i> .....	73
JCAL 20-Pin Header Connector .....	74
<b>DECLARATION OF CONFORMITY .....</b>	<b>75</b>
<b>APPENDIX A: E-POINT JUMPERS .....</b>	<b>76</b>
<b>APPENDIX B: SCHEMATICS .....</b>	<b>77</b>
<b>APPENDIX C: USING POINTERS.....</b>	<b>78</b>
Manual ADC Read Using Pointers .....	78

## INTRODUCTION

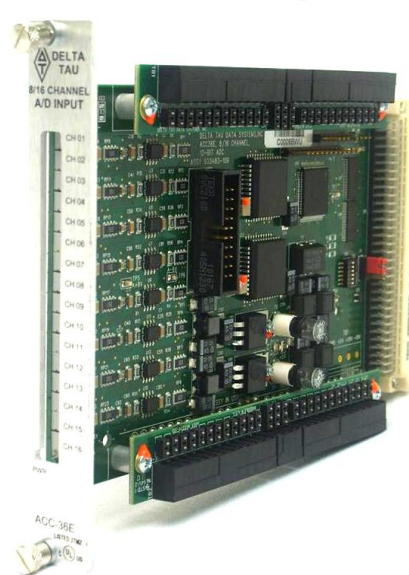
---

The accessory 36E (ACC-36E) is a 16-channel (12-bit) analog data acquisition board capable of converting 16 analog inputs to digital signals. It is offered with either Terminal block or D-Sub connectors.

**ACC-36E:  
D-Sub Option**



**ACC-36E:  
Terminal Block Option**



## SPECIFICATIONS

### Environmental Specifications

---

Description	Specification	Notes
Operating Temperature	0 °C to 45 °C	
Storage Temperature	-25 °C to 70 °C	
Humidity	10% to 95%	Non-Condensing

### Electrical Specifications

---

#### Power Requirements

Whether providing the ACC-36E with power from the 3U backplane bus or externally (standalone mode) through TB1, the power requirements ( $\pm 10\%$ ) are:

+ 5 V @ 150 mA

+15 V @ 20 mA

- 15 V @ 20 mA

#### ACC-36E Fuse

Manufacturer	Specification	Delta Tau Part Number
Little Fuse	125 V @ 2.0 A	273.500

#### ADC Bandwidth

The cutoff frequency of the ADC inputs is approximately 10 KHz.

#### Agency Approval and Safety

Item	Description
CE Mark	Full Compliance
EMC	EN55011 Class A Group 1 EN61000-3-2 Class A EN61000-3-3 EN61000-4-2 EN61000-4-3 EN61000-4-4 EN61000-4-5 EN61000-4-6 EN61000-4-11
Safety	EN 61010-1
UL	UL 61010-1 File E314517
cUL	CAN/CSA C22.2 No. 1010.1-92 File E314517
Flammability Class	UL 94V-0



## Adjustment Potentiometers

ADC Channel	ADC Potentiometer
1	R8
2	R16
3	R24
4	R32
5	R40
6	R48
7	R58
8	R67
9	R7
10	R15
11	R23
12	R31
13	R39
14	R47
15	R58
16	R66



### Note

- R49 and R65 are for reference voltages adjustment on ADC chips
- R50 and R68 are for digital offset adjustment on ADC chips
- These pots should not be adjusted by users

## ADC Chips

The Analog-to-Digital Converter (ADC) units used in ACC-36E are the MAX180 monolithic devices manufactured by Maxim Integrated Products. These devices have 12-bit resolution with  $\pm 1/2$  Least Significant Bit (LSB) linearity specification. For more details on the ADC chips, refer to the data sheet published by the manufacturer:

Document 19-3950; Rev 0, 6/91  
Complete, 8-Channel, 12-Bit Data Acquisition Systems  
Maxim Integrated Products  
120 San Gabriel Drive  
Sunnyvale, CA 94086  
Phone: (408) 737-7600

## Physical Specifications

---

Description	Specification	Notes
Terminal Block Connectors	FRONT-MC1,5/12-ST3,81 FRONT-MC1,5/5-ST3,81 FRONT-MC1,5/3-ST3,81	Terminal Blocks from Phoenix Contact (UL-94V0)
DB Option Connectors	DB15 Female	UL-94V0

## ADDRESSING ACC-36E

Dip switch (SW1) specifies the base address of the ACC-36E in a 3U TURBO / POWER UMAC, or MACRO Station rack.

### Turbo/Power UMAC, MACRO Station Dip Switch Settings

Chip Select	Base Address				SW1 Positions					
	TURBO	MACRO	POWER		6	5	4	3	2	1
			Offset	Index (n)						
CS10	Y:\$78C00	Y:\$8800	\$A00000	0	ON	ON	ON	ON	ON	ON
	Y:\$79C00	Y:\$9800	\$A08000	4	ON	ON	ON	OFF	ON	ON
	Y:\$7AC00	Y:\$A800	\$A10000	8	ON	ON	OFF	ON	ON	ON
	Y:\$7BC00	Y:\$B800	\$A18000	12	ON	ON	OFF	OFF	ON	ON
CS12	Y:\$78D00	Y:\$8840	\$B00000	1	ON	ON	ON	ON	ON	OFF
	Y:\$79D00	Y:\$9840	\$B08000	5	ON	ON	ON	OFF	ON	OFF
	Y:\$7AD00	Y:\$A840	\$B10000	9	ON	ON	OFF	ON	ON	OFF
	Y:\$7BD00	Y:\$B840	\$B18000	13	ON	ON	OFF	OFF	ON	OFF
CS14	Y:\$78E00	Y:\$8880	\$C00000	2	ON	ON	ON	ON	OFF	ON
	Y:\$79E00	Y:\$9880	\$C08000	6	ON	ON	ON	OFF	OFF	ON
	Y:\$7AE00	Y:\$A880	\$C10000	10	ON	ON	OFF	ON	OFF	ON
	Y:\$7BE00	Y:\$B880	\$C18000	14	ON	ON	OFF	OFF	OFF	ON
CS16	Y:\$78F00	Y:\$88C0	\$D00000	3	ON	ON	ON	ON	OFF	OFF
	Y:\$79F00	Y:\$98C0	\$D08000	7	ON	ON	ON	OFF	OFF	OFF
	Y:\$7AF00	Y:\$A8C0	\$D10000	11	ON	ON	OFF	ON	OFF	OFF
	Y:\$7BF00	Y:\$B8C0	\$D18000	15	ON	ON	OFF	OFF	OFF	OFF



*Note*

- ON designates Closed. OFF designates Open
- Factory default is all ON
- The maximum addressable number of ACC-36Es (or similar type accessories) in a single rack is 16

### Legacy MACRO Dip Switch Settings

Chip Select	Base Address (Alternate)	SW1 Positions					
		6	5	4	3	2	1
10	Y:\$B800 (Y:\$FFE0)	ON (OFF)	ON (OFF)	OFF	OFF	ON	ON
12	Y:\$B840 (Y:\$FFE8)	ON (OFF)	ON (OFF)	OFF	OFF	ON	OFF
14	Y:\$B880 (Y:\$FFF0)	ON (OFF)	ON (OFF)	OFF	OFF	OFF	ON
16	Y:\$B8C0 (Y:\$FFF8)	ON (OFF)	ON (OFF)	ON	ON	OFF	OFF



*Note*

The Legacy Macro base addresses are double mapped. Set SW1 positions 5 & 6 to OFF if the alternate addressing is desired.

## Hardware Address Limitations

Two types of accessory cards have been designed for the UMAC 3U bus type rack; type A and type B cards. They can be sorted out as follows:

Name	Type	Category	Possible Number of Addresses	Maximum Number of cards in 1 rack
ACC-9E	A	General I/O	4	12
ACC-10E	A	General I/O	4	
ACC-11E	A	General I/O	4	
ACC-12E	A	General I/O	4	
ACC-14E	B	General I/O	16	16
ACC-28E	B	Analog I/O	16	
ACC-36E	B	Analog I/O	16	
ACC-53E	B	Feedback	16	
ACC-57E	B	Feedback	16	
ACC-58E	B	Feedback	16	
ACC-59E	B	Analog I/O	12	12
ACC-65E	B	General I/O	16	16
ACC-66E	B	General I/O	16	
ACC-67E	B	General I/O	16	
ACC-68E	B	General I/O	16	
ACC-84E	B	Feedback	12	12

Addressing Type A and Type B accessory cards in a UMAC or MACRO station rack requires attention to the following set of rules:

### Populating Rack with Type A Cards Only (no conflicts)

In this mode, the card(s) can potentially use any available Address/Chip Select.



**Note**

Type A cards can have up to 4 different base addresses. Because each card can be set up (jumper settings) to use the lower, middle or higher byte of a specific base address, it is possible to populate a single rack with a maximum of 12 Type A accessory cards.

### Populating Rack with Type B Cards Only (no conflicts)

In this mode, the card(s) can potentially use any available Address/Chip Select.

### Populating Rack with Type A & Type B Cards (possible conflicts)

- Typically, Type A and Type B cards should not share the same Chip Select. If this configuration is possible, then the next couple of rules does not apply, and can be disregarded.
- Type A cards cannot share the same Chip Select as Type B Feedback cards.
- Type A cards can share the same Chip Select as Type B general I/O cards. However, in this mode Type B cards naturally use the lower byte (default), and Type A cards must be set up (jumper settings) to the middle/high byte of the selected base address.

## USING ACC-36E WITH TURBO UMAC

---

### Setting Up the Analog Inputs (ADCs)

---

The A/D converter chips used on the ACC-36E multiplex the resulting data, and therefore it is mandatory to read each input one at a time. With Turbo UMAC, this can be done in two ways:

- Automatic Read
- Manual (semi-automatic) Read



The automatic read feature is available with Turbo firmware version 1.936 or newer. It supports up to two ACC-36Es in one rack.

*Note*

---

## Automatic ADC Read

The automatic read function demultiplexes the data into individual registers and stores them in accessible memory locations. It can handle up to a total of 32 ADC channels. There are 16 ADCs (8 pairs) per base address (or ACC-36E card) for a maximum of 2 base addresses (or two ACC-36E cards).

Pair #	1 <sup>st</sup> ACC-36E	Pair #	2 <sup>nd</sup> ACC-36E
1	ADC#1 & ADC#9	9	ADC#1 & ADC#9
2	ADC#2 & ADC#10	10	ADC#2 & ADC#10
3	ADC#3 & ADC#11	11	ADC#3 & ADC#11
4	ADC#4 & ADC#12	12	ADC#4 & ADC#12
5	ADC#5 & ADC#13	13	ADC#5 & ADC#13
6	ADC#6 & ADC#14	14	ADC#6 & ADC#14
7	ADC#7 & ADC#15	15	ADC#7 & ADC#15
8	ADC#8 & ADC#16	16	ADC#8 & ADC#16

These are the necessary steps for setting up the automatic read function:

1. **Configure A/D Processing Ring Size;** I5060. This is the number of ADC pairs to be demuxed.  
I5060 = Number of ADC Pairs to Sample



*Note*

- Saving I5060 to a value greater than zero, the A/D Convert Enable I5080 is automatically set to 1 on after **Save** and \$\$\$ or power-up. Subsequently setting I5080 to 0, the user can suspend the de-multiplexing, to be resumed by setting I5080=1.
- Each ADC is automatically updated every I5060 phase cycles.

2. **Configure A/D Ring Pointers;** I5061 thru I5076.  
I5061 through I5068 specify the offset widths (hex) of the first 8 pairs (1<sup>st</sup> ACC-36E).  
I5069 through I5076 specify the offset widths (hex) of the second 8 pairs (2<sup>nd</sup> ACC-36E).  
\$(Offset Width) = \$(Card Base Address) – \$078800
3. **Configure A/D Ring Convert Codes;** I5081 thru I5096.  
I5081 through I5088 define the voltage modes of the first 8 pairs (1<sup>st</sup> ACC-36E).  
I5089 through I5096 define the voltage modes of the second 8 pairs (2<sup>nd</sup> ACC-36E).  
Refer to the Table of Automatic Read Convert Code shown on the next page for these codes.
4. Assign M-Variables to demuxed data registers, then **Save** and reset (\$\$\$).

## Automatic ADC Read Example 1

Setting up Turbo UMAC with an ACC-36E at base address \$078C00 to automatically read all 16 ADCs, allowing the user to choose unipolar or bipolar mode:

### 1. A/D Processing Ring Size

I5060=8	; Demux 8 ADC pairs
---------	---------------------

### 2. A/D Ring Pointers

I5061,8=\$400	; ADC pairs 1 through 8 Offset width \$400 = \$078C00-\$78800
---------------	---

### 3. A/D Convert Code

Pair #	Unipolar	Bipolar
1	I5081=0 ; ADC#1 & #9 Unipolar	I5081=8 ; ADC#1 & #9 Bipolar
2	I5082=1 ; ADC#2 & #10 Unipolar	I5082=9 ; ADC#2 & #10 Bipolar
3	I5083=2 ; ADC#3 & #11 Unipolar	I5083=10 ; ADC#3 & #11 Bipolar
4	I5084=3 ; ADC#4 & #12 Unipolar	I5084=11 ; ADC#4 & #12 Bipolar
5	I5085=4 ; ADC#5 & #13 Unipolar	I5085=12 ; ADC#5 & #13 Bipolar
6	I5086=5 ; ADC#6 & #14 Unipolar	I5086=13 ; ADC#6 & #14 Bipolar
7	I5087=6 ; ADC#7 & #15 Unipolar	I5087=14 ; ADC#7 & #15 Bipolar
8	I5088=7 ; ADC#8 & #16 Unipolar	I5088=15 ; ADC#8 & #16 Bipolar

### 4. Assigning M-Variables to Demuxed Data Registers

Unipolar	Bipolar
M5061->Y:\$003400,12,12,U ;ADC# 1 Unipolar	M5061->Y:\$003400,12,12,S ;ADC# 1 Bipolar
M5062->Y:\$003402,12,12,U ;ADC# 2 Unipolar	M5062->Y:\$003402,12,12,S ;ADC# 2 Bipolar
M5063->Y:\$003404,12,12,U ;ADC# 3 Unipolar	M5063->Y:\$003404,12,12,S ;ADC# 3 Bipolar
M5064->Y:\$003406,12,12,U ;ADC# 4 Unipolar	M5064->Y:\$003406,12,12,S ;ADC# 4 Bipolar
M5065->Y:\$003408,12,12,U ;ADC# 5 Unipolar	M5065->Y:\$003408,12,12,S ;ADC# 5 Bipolar
M5066->Y:\$00340A,12,12,U ;ADC# 6 Unipolar	M5066->Y:\$00340A,12,12,S ;ADC# 6 Bipolar
M5067->Y:\$00340C,12,12,U ;ADC# 7 Unipolar	M5067->Y:\$00340C,12,12,S ;ADC# 7 Bipolar
M5068->Y:\$00340E,12,12,U ;ADC# 8 Unipolar	M5068->Y:\$00340E,12,12,S ;ADC# 8 Bipolar
M5069->Y:\$003401,12,12,U ;ADC# 9 Unipolar	M5069->Y:\$003401,12,12,S ;ADC# 9 Bipolar
M5070->Y:\$003403,12,12,U ;ADC#10 Unipolar	M5070->Y:\$003403,12,12,S ;ADC#10 Bipolar
M5071->Y:\$003405,12,12,U ;ADC#11 Unipolar	M5071->Y:\$003405,12,12,S ;ADC#11 Bipolar
M5072->Y:\$003407,12,12,U ;ADC#12 Unipolar	M5072->Y:\$003407,12,12,S ;ADC#12 Bipolar
M5073->Y:\$003409,12,12,U ;ADC#13 Unipolar	M5073->Y:\$003409,12,12,S ;ADC#13 Bipolar
M5074->Y:\$00340B,12,12,U ;ADC#14 Unipolar	M5074->Y:\$00340B,12,12,S ;ADC#14 Bipolar
M5075->Y:\$00340D,12,12,U ;ADC#15 Unipolar	M5075->Y:\$00340D,12,12,S ;ADC#15 Bipolar
M5076->Y:\$00340F,12,12,U ;ADC#16 Unipolar	M5076->Y:\$00340F,12,12,S ;ADC#16 Bipolar



*Note*

- Issue a **Save** and reset (\$\$\$) after download to enable the A/D ring processing
- ADC channels that come in pairs cannot have different convert codes; e.g., if ADC#1 is unipolar, ADC#9 cannot be bipolar
- If the convert code is unipolar, the M-Variable assigned to read the ADC result must also be unipolar (unsigned)

## Automatic ADC Read Example 2

Another ACC-36E has been added to a total of two cards respectively at \$078C00 and \$079C00:

## 1. A/D Processing Ring Size and Ring Pointers

I5060=16	; Demux 16 ADC pairs	
I5061,8=\$400	; ADC pairs 1 thru 8 offset width	\$400 = \$078C00 - \$078800
I5069,8=\$1400	; ADC pairs 9 thru 16 offset width	\$1400 = \$079C00 - \$078800

## 2. A/D Convert Codes

	Unipolar	Bipolar
1 <sup>st</sup> ACC-36E	I5081=0 ; ADC#1 & #9 Unipolar	I5081=8 ; ADC#1 & #9 Bipolar
	I5082=1 ; ADC#2 & #10 Unipolar	I5082=9 ; ADC#2 & #10 Bipolar
	I5083=2 ; ADC#3 & #11 Unipolar	I5083=10 ; ADC#3 & #11 Bipolar
	I5084=3 ; ADC#4 & #12 Unipolar	I5084=11 ; ADC#4 & #12 Bipolar
	I5085=4 ; ADC#5 & #13 Unipolar	I5085=12 ; ADC#5 & #13 Bipolar
	I5086=5 ; ADC#6 & #14 Unipolar	I5086=13 ; ADC#6 & #14 Bipolar
	I5087=6 ; ADC#7 & #15 Unipolar	I5087=14 ; ADC#7 & #15 Bipolar
	I5088=7 ; ADC#8 & #16 Unipolar	I5088=15 ; ADC#8 & #16 Bipolar
2 <sup>nd</sup> ACC-36E	I5089=0 ; ADC#1 & #9 Unipolar	I5089=8 ; ADC#1 & #9 Bipolar
	I5090=1 ; ADC#2 & #10 Unipolar	I5090=9 ; ADC#2 & #10 Bipolar
	I5091=2 ; ADC#3 & #11 Unipolar	I5091=10 ; ADC#3 & #11 Bipolar
	I5092=3 ; ADC#4 & #12 Unipolar	I5092=11 ; ADC#4 & #12 Bipolar
	I5093=4 ; ADC#5 & #13 Unipolar	I5093=12 ; ADC#5 & #13 Bipolar
	I5094=5 ; ADC#6 & #14 Unipolar	I5094=13 ; ADC#6 & #14 Bipolar
	I5095=6 ; ADC#7 & #15 Unipolar	I5095=14 ; ADC#7 & #15 Bipolar
	I5096=7 ; ADC#8 & #16 Unipolar	I5096=15 ; ADC#8 & #16 Bipolar

## 3. Assigning M-Variables to Demuxed Data Registers

	Unipolar	Bipolar
1 <sup>st</sup> ACC-36E	M5061->Y:\$003400,12,12,U ; ADC#1	M5061->Y:\$003400,12,12,S ; ADC#1
	M5062->Y:\$003402,12,12,U ; ADC#2	M5062->Y:\$003402,12,12,S ; ADC#2
	M5063->Y:\$003404,12,12,U ; ADC#3	M5063->Y:\$003404,12,12,S ; ADC#3
	M5064->Y:\$003406,12,12,U ; ADC#4	M5064->Y:\$003406,12,12,S ; ADC#4
	M5065->Y:\$003408,12,12,U ; ADC#5	M5065->Y:\$003408,12,12,S ; ADC#5
	M5066->Y:\$00340A,12,12,U ; ADC#6	M5066->Y:\$00340A,12,12,S ; ADC#6
	M5067->Y:\$00340C,12,12,U ; ADC#7	M5067->Y:\$00340C,12,12,S ; ADC#7
	M5068->Y:\$00340E,12,12,U ; ADC#8	M5068->Y:\$00340E,12,12,S ; ADC#8
	M5069->Y:\$003401,12,12,U ; ADC#9	M5069->Y:\$003401,12,12,S ; ADC#9
	M5070->Y:\$003403,12,12,U ; ADC#10	M5070->Y:\$003403,12,12,S ; ADC#10
	M5071->Y:\$003405,12,12,U ; ADC#11	M5071->Y:\$003405,12,12,S ; ADC#11
	M5072->Y:\$003407,12,12,U ; ADC#12	M5072->Y:\$003407,12,12,S ; ADC#12
	M5073->Y:\$003409,12,12,U ; ADC#13	M5073->Y:\$003409,12,12,S ; ADC#13
	M5074->Y:\$00340B,12,12,U ; ADC#14	M5074->Y:\$00340B,12,12,S ; ADC#14
	M5075->Y:\$00340D,12,12,U ; ADC#15	M5075->Y:\$00340D,12,12,S ; ADC#15
	M5076->Y:\$00340F,12,12,U ; ADC#16	M5076->Y:\$00340F,12,12,S ; ADC#16
2 <sup>nd</sup> ACC-36E	M5161->Y:\$003410,12,12,U ; ADC#1	M5161->Y:\$003410,12,12,S ; ADC#1
	M5162->Y:\$003412,12,12,U ; ADC#2	M5162->Y:\$003412,12,12,S ; ADC#2
	M5163->Y:\$003414,12,12,U ; ADC#3	M5163->Y:\$003414,12,12,S ; ADC#3
	M5164->Y:\$003416,12,12,U ; ADC#4	M5164->Y:\$003416,12,12,S ; ADC#4
	M5165->Y:\$003418,12,12,U ; ADC#5	M5165->Y:\$003418,12,12,S ; ADC#5
	M5166->Y:\$00341A,12,12,U ; ADC#6	M5166->Y:\$00341A,12,12,S ; ADC#6
	M5167->Y:\$00341C,12,12,U ; ADC#7	M5167->Y:\$00341C,12,12,S ; ADC#7
	M5168->Y:\$00341E,12,12,U ; ADC#8	M5168->Y:\$00341E,12,12,S ; ADC#8
	M5169->Y:\$003411,12,12,U ; ADC#9	M5169->Y:\$003411,12,12,S ; ADC#9
	M5170->Y:\$003413,12,12,U ; ADC#10	M5170->Y:\$003413,12,12,S ; ADC#10
	M5171->Y:\$003415,12,12,U ; ADC#11	M5171->Y:\$003415,12,12,S ; ADC#11
	M5172->Y:\$003417,12,12,U ; ADC#12	M5172->Y:\$003417,12,12,S ; ADC#12
	M5173->Y:\$003419,12,12,U ; ADC#13	M5173->Y:\$003419,12,12,S ; ADC#13
	M5174->Y:\$00341B,12,12,U ; ADC#14	M5174->Y:\$00341B,12,12,S ; ADC#14
	M5175->Y:\$00341D,12,12,U ; ADC#15	M5175->Y:\$00341D,12,12,S ; ADC#15
	M5176->Y:\$00341F,12,12,U ; ADC#16	M5176->Y:\$00341F,12,12,S ; ADC#16

## Manual ADC Read

The manual read method consists of choosing the desired channel with a pointer, reading it, waiting for the conversion to finish, and then copying the contents of the result register into a UMAC memory location.

The following are the necessary steps for implementing the manual ADC read method. The example parameters given here are for an ACC-36E at base address \$078C00, allowing the user to choose unipolar or bipolar input signals:

1. Point one available M-Variable (12-bit wide) to the lower or higher 12 bits of the ACC-36E base address depending on which ADC channel data is desired:

ADC#1 through ADC#8 are in bits 0-11

ADC#9 through ADC#16 are in bits 12-23

These are the “**Data Read**” registers of the selected channel; they can be defined as:

	Unipolar	Bipolar
<b>Lower 12 bits</b>	M5000->Y:\$078C00,0,12,U	M5001->Y:\$078C00,0,12,S
<b>Upper 12 bits</b>	M5002->Y:\$078C00,12,12,U	M5003->Y:\$078C00,12,12,S

2. Point an available M-Variable (24-bit wide unsigned) to the base address of the ACC-36E. This is the “**Channel Select**” Pointer.

```
M5004->Y:$078C00,0,24,U
```

3. Point an M-Variable to the bits that indicate when the lower and upper ADCs have finished their conversions. These are the low and high **ADC Ready Bits**, respectively. They are 0 while the conversion is in process and then become 1 when the ADC conversion has finished.

```
M5005->Y:$078F30,5,1 // Low ADC Ready Bit (Channels 1-8)
M5006->Y:$078F31,5,1 // High ADC Ready Bit (Channels 9-16)
```

The addresses of these bits are based on the card base address as follows:

Base Address	Low ADC Ready Bit	High ADC Ready Bit
Y:\$78C00	M5005->Y:\$078F30,5,1	M5006->Y:\$078F31,5,1
Y:\$79C00	M5005->Y:\$078F34,5,1	M5006->Y:\$078F35,5,1
Y:\$7AC00	M5005->Y:\$078F38,5,1	M5006->Y:\$078F39,5,1
Y:\$7BC00	M5005->Y:\$078F3C,5,1	M5006->Y:\$078F3D,5,1
Y:\$78D00	M5005->Y:\$079F30,5,1	M5006->Y:\$079F31,5,1
Y:\$79D00	M5005->Y:\$079F34,5,1	M5006->Y:\$079F35,5,1
Y:\$7AD00	M5005->Y:\$079F38,5,1	M5006->Y:\$079F39,5,1
Y:\$7BD00	M5005->Y:\$079F3C,5,1	M5006->Y:\$079F3D,5,1
Y:\$78E00	M5005->Y:\$07AF30,5,1	M5006->Y:\$07AF31,5,1
Y:\$79E00	M5005->Y:\$07AF34,5,1	M5006->Y:\$07AF35,5,1
Y:\$7AE00	M5005->Y:\$07AF38,5,1	M5006->Y:\$07AF39,5,1
Y:\$7BE00	M5005->Y:\$07AF3C,5,1	M5006->Y:\$07AF3D,5,1
Y:\$78F00	M5005->Y:\$07BF30,5,1	M5006->Y:\$07BF31,5,1
Y:\$79F00	M5005->Y:\$07BF34,5,1	M5006->Y:\$07BF35,5,1
Y:\$7AF00	M5005->Y:\$07BF38,5,1	M5006->Y:\$07BF39,5,1
Y:\$7BF00	M5005->Y:\$07BF3C,5,1	M5006->Y:\$07BF3D,5,1



4. Using the **Channel Select** Pointer, specify the channel # and voltage mode as follows:

ADC Channel	Data Read	Channel Select Value	
ADC# Read by M5058	ADC# Read by M5059	Unipolar	Bipolar
ADC#1	ADC#9	0	8
ADC#2	ADC#10	1	9
ADC#3	ADC#11	2	10
ADC#4	ADC#12	3	11
ADC#5	ADC#13	4	12
ADC#6	ADC#14	5	13
ADC#7	ADC#15	6	14
ADC#8	ADC#16	7	15

5. Wait for the **ADC Ready** bits to become 1, then read and/or copy data from the **Data Read** M-Variables.

**Note**

These are only suggested definitions for the Data Read and Channel Select pointers. The user can choose whichever M-Variable numbering he or she desires.

---

## ADC Manual Read Example PLCs

Ultimately, the procedure above can be implemented in a PLC script to read all channels consecutively and consistently, creating a “custom automatic” function, as seen in the following examples.

### Unipolar PLC Example

Setting up Turbo UMAC with an ACC-36E at base address \$078C00 with all 16 ADCs as unipolar. This example uses M5000, M5002, M5004–M5006, and P2001–P2016.

```
#define DataRead_Low      M5000    ; Lower 12-bit Data Read register
#define DataRead_High    M5002    ; Higher 12-bit Data Read register
#define ChSelect          M5004    ; Channel Select Pointer
#define LowADCReady       M5005    ; Low ADC Ready Bit
#define HighADCReady      M5006    ; High ADC Ready Bit
DataRead_Low->Y:$078C00,0,12,U    ; Unsigned for unipolar
DataRead_High->Y:$078C00,12,12,U  ; Unsigned for unipolar
ChSelect->Y:$078C00,0,24,U        ; Channel Select
LowADCReady->Y:$078F30,5,1        ; Low ADC Ready Bit
HighADCReady->Y:$078F31,5,1       ; High ADC Ready Bit
#define ADC1              P2001    ; Channel 1 ADC storage
#define ADC2              P2002    ; Channel 2 ADC storage
#define ADC3              P2003    ; Channel 3 ADC storage
#define ADC4              P2004    ; Channel 4 ADC storage
#define ADC5              P2005    ; Channel 5 ADC storage
#define ADC6              P2006    ; Channel 6 ADC storage
#define ADC7              P2007    ; Channel 7 ADC storage
#define ADC8              P2008    ; Channel 8 ADC storage
#define ADC9              P2009    ; Channel 9 ADC storage
#define ADC10             P2010    ; Channel 10 ADC storage
#define ADC11             P2011    ; Channel 11 ADC storage
#define ADC12             P2012    ; Channel 12 ADC storage
#define ADC13             P2013    ; Channel 13 ADC storage
#define ADC14             P2014    ; Channel 14 ADC storage
#define ADC15             P2015    ; Channel 15 ADC storage
#define ADC16             P2016    ; Channel 16 ADC storage

Open PLC 1 Clear
ChSelect=0                      ; Select Channels 1 and 9 (unipolar)
While(LowADCReady != 1 and HighADCReady != 1) EndWhile ; Wait for ADC Conversions to Finish
ADC1=DataRead_Low                ; Read/Copy result into storage
ADC9=DataRead_High               ; Read/Copy result into storage
ChSelect=1                      ; Select Channels 2 and 10 (unipolar)
While(LowADCReady != 1 and HighADCReady != 1) EndWhile ; Wait for ADC Conversions to Finish
ADC2=DataRead_Low                ; Read/Copy result into storage
ADC10=DataRead_High              ; Read/Copy result into storage
ChSelect=2                      ; Select Channels 3 and 11 (unipolar)
While(LowADCReady != 1 and HighADCReady != 1) EndWhile ; Wait for ADC Conversions to Finish
ADC3=DataRead_Low                ; Read/Copy result into storage
ADC11=DataRead_High              ; Read/Copy result into storage
ChSelect=3                      ; Select Channels 4 and 12 (unipolar)
While(LowADCReady != 1 and HighADCReady != 1) EndWhile ; Wait for ADC Conversions to Finish
ADC4=DataRead_Low                ; Read/Copy result into storage
ADC12=DataRead_High              ; Read/Copy result into storage
ChSelect=4                      ; Select Channels 5 and 13 (unipolar)
While(LowADCReady != 1 and HighADCReady != 1) EndWhile ; Wait for ADC Conversions to Finish
ADC5=DataRead_Low                ; Read/Copy result into storage
ADC13=DataRead_High              ; Read/Copy result into storage
ChSelect=5                      ; Select Channels 6 and 14 (unipolar)
While(LowADCReady != 1 and HighADCReady != 1) EndWhile ; Wait for ADC Conversions to Finish
ADC6=DataRead_Low                ; Read/Copy result into storage
ADC14=DataRead_High              ; Read/Copy result into storage
ChSelect=6                      ; Select Channels 7 and 15 (unipolar)
While(LowADCReady != 1 and HighADCReady != 1) EndWhile ; Wait for ADC Conversions to Finish
ADC7=DataRead_Low                ; Read/Copy result into storage
ADC15=DataRead_High              ; Read/Copy result into storage
ChSelect=7                      ; Select Channels 8 and 16 (unipolar)
While(LowADCReady != 1 and HighADCReady != 1) EndWhile ; Wait for ADC Conversions to Finish
ADC8=DataRead_Low                ; Read/Copy result into storage
ADC16=DataRead_High              ; Read/Copy result into storage
Close
```

**Bipolar PLC Example**

Setting up Turbo UMAC with an ACC-36E at base address \$078C00 with all 16 ADCs as bipolar. This example uses M5001, M5003–M5006, and P2001–P2016.

```
#define DataRead_Low      M5001    ; Lower 12-bit Data Read register
#define DataRead_High    M5003    ; Higher 12-bit Data Read register
#define ChSelect          M5004    ; Channel Select Pointer
#define LowADCReady       M5005    ; Low ADC Ready Bit
#define HighADCReady      M5006    ; High ADC Ready Bit
DataRead_Low->Y:$078C00,0,12,S    ; Signed for bipolar
DataRead_High->Y:$078C00,12,12,S  ; Signed for bipolar
ChSelect->Y:$078C00,0,24,U        ; Channel Select
LowADCReady->Y:$078F30,5,1        ; Low ADC Ready Bit
HighADCReady->Y:$078F31,5,1       ; High ADC Ready Bit
#define ADC1              P2001    ; Channel 1 ADC storage
#define ADC2              P2002    ; Channel 2 ADC storage
#define ADC3              P2003    ; Channel 3 ADC storage
#define ADC4              P2004    ; Channel 4 ADC storage
#define ADC5              P2005    ; Channel 5 ADC storage
#define ADC6              P2006    ; Channel 6 ADC storage
#define ADC7              P2007    ; Channel 7 ADC storage
#define ADC8              P2008    ; Channel 8 ADC storage
#define ADC9              P2009    ; Channel 9 ADC storage
#define ADC10             P2010    ; Channel 10 ADC storage
#define ADC11             P2011    ; Channel 11 ADC storage
#define ADC12             P2012    ; Channel 12 ADC storage
#define ADC13             P2013    ; Channel 13 ADC storage
#define ADC14             P2014    ; Channel 14 ADC storage
#define ADC15             P2015    ; Channel 15 ADC storage
#define ADC16             P2016    ; Channel 16 ADC storage

Open PLC 2 Clear
ChSelect=8                    ; Select Channels 1 and 9 (bipolar)
While(LowADCReady != 1 and HighADCReady != 1) EndWhile ; Wait for ADC Conversions to Finish
ADC1=DataRead_Low             ; Read/Copy result into storage
ADC9=DataRead_High            ; Read/Copy result into storage
ChSelect=9                    ; Select Channels 2 and 10 (bipolar)
While(LowADCReady != 1 and HighADCReady != 1) EndWhile ; Wait for ADC Conversions to Finish
ADC2=DataRead_Low             ; Read/Copy result into storage
ADC10=DataRead_High           ; Read/Copy result into storage
ChSelect=10                   ; Select Channels 3 and 11 (bipolar)
While(LowADCReady != 1 and HighADCReady != 1) EndWhile ; Wait for ADC Conversions to Finish
ADC3=DataRead_Low             ; Read/Copy result into storage
ADC11=DataRead_High           ; Read/Copy result into storage
ChSelect=11                   ; Select Channels 4 and 12 (bipolar)
While(LowADCReady != 1 and HighADCReady != 1) EndWhile ; Wait for ADC Conversions to Finish
ADC4=DataRead_Low             ; Read/Copy result into storage
ADC12=DataRead_High           ; Read/Copy result into storage
ChSelect=12                   ; Select Channels 5 and 13 (bipolar)
While(LowADCReady != 1 and HighADCReady != 1) EndWhile ; Wait for ADC Conversions to Finish
ADC5=DataRead_Low             ; Read/Copy result into storage
ADC13=DataRead_High           ; Read/Copy result into storage
ChSelect=13                   ; Select Channels 6 and 14 (bipolar)
While(LowADCReady != 1 and HighADCReady != 1) EndWhile ; Wait for ADC Conversions to Finish
ADC6=DataRead_Low             ; Read/Copy result into storage
ADC14=DataRead_High           ; Read/Copy result into storage
ChSelect=14                   ; Select Channels 7 and 15 (bipolar)
While(LowADCReady != 1 and HighADCReady != 1) EndWhile ; Wait for ADC Conversions to Finish
ADC7=DataRead_Low             ; Read/Copy result into storage
ADC15=DataRead_High           ; Read/Copy result into storage
ChSelect=15                   ; Select Channels 8 and 16 (bipolar)
While(LowADCReady != 1 and HighADCReady != 1) EndWhile ; Wait for ADC Conversions to Finish
ADC8=DataRead_Low             ; Read/Copy result into storage
ADC16=DataRead_High           ; Read/Copy result into storage
Close
```

## Testing the Analog Inputs

The Analog Inputs can be brought into the ACC-36E as single ended (ADC+ & Ground) or differential (ADC+ & ADC-) signals.



### *Note*

In single-ended mode, ADC- should to be tied to analog ground for full resolution and proper operation.

Reading the input signals in software counts using the predefined M-Variables should show the following:

ADC Input	Single-Ended [V] ADC+ <=> AGND	Differential [V] ADC+ <=> ADC-	Software Counts
Unipolar Mode	0	0	0
	10	10	2047
	20	20	4095
Bipolar Mode	-10	-10	-2047
	0	0	0
	10	10	2047

## Using an Analog Input for Servo Feedback

The ACC-36E analog inputs can be used as a feedback device for a servo motor.



*Note*

- Refer to Delta Tau's released application notes or Turbo User Manual for cascaded-loop control (i.e. force, height control around position loop).
- The automatic ADC read function is recommended for this application.

### Example:

Setting up Motor #1 position and velocity feedback to ADC channel #1.

The analog input is brought into the Encoder Conversion Table as a Parallel Y word with no filtering:

The equivalent code in Turbo PMAC Encoder Conversion Table parameters:

```
I8000=$203400 ; Unfiltered parallel pos of location Y:$3400
I8001=$00C00C ; Width and Offset. (Processed Data Location)
```

The position and velocity pointers are then set to the processed data address (i.e. \$3502)

```
I103=$3502 ; Motor #1 position loop feedback address
I104=$3502 ; Motor #1 velocity loop feedback address
```

## Analog Input Power-On Position

Some analog devices are absolute along the travel range of the motor (e.g., in hydraulic piston applications). Generally, it is desirable to obtain the motor position (input voltage) on power up or reset.

This procedure can be done in a simple PLC on power-up by writing the processed A/D data into the motor actual position register (suggested M-Variable Mxx62).



*Note*

- If the automatic ADC read method is being used, waiting a delay of about ½ second after the PMAC boots should be allowed for processing the data before copying it into the motor actual position register.
- If the manual ADC read method is being used, it is recommended to add this procedure at the end of the manual read PLC, or in a subsequent PLC with ½ sec delay to allow data processing.

**Example:** Reading Motor #1 position on power-up or reset, assuming that the automatic read function is used, and that M5061 is predefined to read ADC channel 1.

```
End Gat
Del Gat
Close

#define Mtr1ActPos          M162      ; Motor #1 Actual Position
                                ; Suggested M-Variable units of 1/32*I108
Mtr1ActPos->D:$8B
#define Ch1ADC              M5061    ; Channel 1 ADC

Open PLC 1 Clear
I5111=500*8388608/I10
While (I5111>0) EndWhile          ; ½ sec delay
Mtr1ActPos=Ch1ADC*32*I108          ; Motor #1 Actual Position (scaled to motor counts)
Disable PLC 1                      ; Scan once on power-up or reset
Close
```

## **USING ACC-36E WITH POWER UMAC**

---

### **Setting Up the Analog Inputs (ADCs)**

---

The A/D converter chips used on the ACC-36E multiplex the resulting data, and therefore it is mandatory to read each input one at a time. With Power UMAC, this can be done in two ways:

- Automatic Read
- Manual (semi-automatic) Read

In either method, the user may use either Power PMAC Script or the C programming language.

## Automatic ADC Read

The automatic read function demultiplexes the data into individual registers and stores them once every **AdcDemux.Enable** phase cycles in accessible memory locations. It can handle up to a total of 32 ADC channels (16 pairs). There are 16 ADCs (8 pairs) per base address (or ACC-36E card) for a maximum of 2 base addresses (i.e. two ACC-36E cards) as follows:

Pair #	1 <sup>st</sup> ACC-36E		Pair #	2 <sup>nd</sup> ACC-36E	
1	ADC#1	&	ADC#9	9	ADC#1 & ADC#9
2	ADC#2	&	ADC#10	10	ADC#2 & ADC#10
3	ADC#3	&	ADC#11	11	ADC#3 & ADC#11
4	ADC#4	&	ADC#12	12	ADC#4 & ADC#12
5	ADC#5	&	ADC#13	13	ADC#5 & ADC#13
6	ADC#6	&	ADC#14	14	ADC#6 & ADC#14
7	ADC#7	&	ADC#15	15	ADC#7 & ADC#15
8	ADC#8	&	ADC#16	16	ADC#8 & ADC#16

These are the necessary steps for setting up the automatic read function:

1. **Configure A/D Ring Pointers: `AdcDemux.Address[i]`.**

For each pair with index *i*, **AdcDemux.Address[i]** must be set to the Power PMAC I/O base offset as specified in the [Addressing](#) section of this manual. Typically, pairs 1 – 8 are assigned to the first ACC-36E, pairs 9 – 16 are assigned to the second ACC-36E.

Ring Pointers	Pair #	Ring Pointers	Pair #
<code>AdcDemux.Address[0]</code>	1	<code>AdcDemux.Address[8]</code>	9
<code>AdcDemux.Address[1]</code>	2	<code>AdcDemux.Address[9]</code>	10
<code>AdcDemux.Address[2]</code>	3	<code>AdcDemux.Address[10]</code>	11
<code>AdcDemux.Address[3]</code>	4	<code>AdcDemux.Address[11]</code>	12
<code>AdcDemux.Address[4]</code>	5	<code>AdcDemux.Address[12]</code>	13
<code>AdcDemux.Address[5]</code>	6	<code>AdcDemux.Address[13]</code>	14
<code>AdcDemux.Address[6]</code>	7	<code>AdcDemux.Address[14]</code>	15
<code>AdcDemux.Address[7]</code>	8	<code>AdcDemux.Address[15]</code>	16

2. **Configure A/D Ring Convert Codes: `AdcDemux.ConvertCode[i]`.**

The convert code allows the user to select the input mode, whether unipolar or bipolar, as well as the number of the ADC pair (Hardware Pair#) to sample.

Setup Structures	Definition of n	Mode	Voltage Input
<code>AdcDemux.ConvertCode[i]=\$n00n00</code>	$n = (\text{Hardware Pair \#}) - 1$	Unipolar	Positive Only
	$n = (\text{Hardware Pair \#}) + 7$	Bipolar	Positive/Negative



*Note*

Hardware Pair# in this case assumes per-card pairs (i.e., Hardware Pair# can only run from 1 to 8, because there are only 8 pairs per card, whereas Software Pair# can run from 0 to 15, since those are the indices of the **AdcDemux** structures).



3. **Configure A/D Processing Ring Size: `AdcDemux.Enable`.****`AdcDemux.Enable`** = Number of ADC Pairs to Demux

Setting **`AdcDemux.Enable`** to a value greater than zero activates the automatic ADC read ring.

---

*Note*

---

4. **Access the A/D Results:** The results are stored in the **`AdcDemux.ResultLow[i]`** and **`AdcDemux.ResultHigh[i]`** structures as shown in the following table:

1 <sup>st</sup> ACC-36E		
<i>i</i>	<b><code>ResultLow[i]</code></b>	<b><code>ResultHigh[i]</code></b>
0	ADC#1	ADC#9
1	ADC#2	ADC#10
2	ADC#3	ADC#11
3	ADC#4	ADC#12
4	ADC#5	ADC#13
5	ADC#6	ADC#14
6	ADC#7	ADC#15
7	ADC#8	ADC#16

2 <sup>nd</sup> ACC-36E		
<i>i</i>	<b><code>ResultLow[i]</code></b>	<b><code>ResultHigh[i]</code></b>
8	ADC#1	ADC#9
9	ADC#2	ADC#10
10	ADC#3	ADC#11
11	ADC#4	ADC#12
12	ADC#5	ADC#13
13	ADC#6	ADC#14
14	ADC#7	ADC#15
15	ADC#8	ADC#16



Each ADC pair is automatically updated every **`AdcDemux.Enable`** phase cycles.

---

*Note*

---

## ADC Automatic Read Example 1

Setting up Power UMAC with an ACC-36E at I/O base address offset \$A00000 to automatically read all 16 ADCs, allowing the user to choose unipolar or bipolar mode:

### 1. A/D Ring Pointers:

```
AdcDemux.Address[0] = $A00000;    // ADC Pair #1 (ADC#1 & #9)
AdcDemux.Address[1] = $A00000;    // ADC Pair #2 (ADC#2 & #10)
AdcDemux.Address[2] = $A00000;    // ADC Pair #3 (ADC#3 & #11)
AdcDemux.Address[3] = $A00000;    // ADC Pair #4 (ADC#4 & #12)
AdcDemux.Address[4] = $A00000;    // ADC Pair #5 (ADC#5 & #13)
AdcDemux.Address[5] = $A00000;    // ADC Pair #6 (ADC#6 & #14)
AdcDemux.Address[6] = $A00000;    // ADC Pair #7 (ADC#7 & #15)
AdcDemux.Address[7] = $A00000;    // ADC Pair #8 (ADC#8 & #16)
```

### 2. A/D Convert Codes:

Pair #	Unipolar	Bipolar
1	AdcDemux.ConvertCode[0]=\$000000;	AdcDemux.ConvertCode[0]=\$800800;
2	AdcDemux.ConvertCode[1]=\$100100;	AdcDemux.ConvertCode[1]=\$900900;
3	AdcDemux.ConvertCode[2]=\$200200;	AdcDemux.ConvertCode[2]=\$A00A00;
4	AdcDemux.ConvertCode[3]=\$300300;	AdcDemux.ConvertCode[3]=\$B00B00;
5	AdcDemux.ConvertCode[4]=\$400400;	AdcDemux.ConvertCode[4]=\$C00C00;
6	AdcDemux.ConvertCode[5]=\$500500;	AdcDemux.ConvertCode[5]=\$D00D00;
7	AdcDemux.ConvertCode[6]=\$600600;	AdcDemux.ConvertCode[6]=\$E00E00;
8	AdcDemux.ConvertCode[7]=\$700700;	AdcDemux.ConvertCode[7]=\$F00F00;

### 3. A/D Processing Ring Size:

```
AdcDemux.Enable = 8;    // Demux 8 ADC pairs
```

### 4. Accessing the A/D Registers:

The resulting data is found in **AdcDemux.ResultLow[i]**, containing the ADC result for the lower-numbered ADC in the pair (e.g. ADC #1 in the pair consisting of ADCs #1 and #9), and **AdcDemux.ResultHigh[i]**, containing the ADC result for the higher-numbered ADC in the pair (e.g. ADC #9 in the pair consisting of ADCs #1 and #9).

## ADC Automatic Read Example 2

Another ACC-36E has been added to produce a total of two cards at base offsets \$A00000 and \$B00000, respectively.

### 1. A/D Processing Ring Size

```
AdcDemux.Enable = 16;           // Demux 16 ADC pairs
```

### 2. A/D Ring Pointers

```
AdcDemux.Address[0] = $A00000;    // ADC Pair #1
AdcDemux.Address[1] = $A00000;    // ADC Pair #2
AdcDemux.Address[2] = $A00000;    // ADC Pair #3
AdcDemux.Address[3] = $A00000;    // ADC Pair #4
AdcDemux.Address[4] = $A00000;    // ADC Pair #5
AdcDemux.Address[5] = $A00000;    // ADC Pair #6
AdcDemux.Address[6] = $A00000;    // ADC Pair #7
AdcDemux.Address[7] = $A00000;    // ADC Pair #8
AdcDemux.Address[8] = $B00000;    // ADC Pair #9
AdcDemux.Address[9] = $B00000;    // ADC Pair #10
AdcDemux.Address[10] = $B00000;   // ADC Pair #11
AdcDemux.Address[11] = $B00000;   // ADC Pair #12
AdcDemux.Address[12] = $B00000;   // ADC Pair #13
AdcDemux.Address[13] = $B00000;   // ADC Pair #14
AdcDemux.Address[14] = $B00000;   // ADC Pair #15
AdcDemux.Address[15] = $B00000;   // ADC Pair #16
```

### 3. A/D Convert Codes

	Pair #	Unipolar	Bipolar
1 <sup>st</sup> ACC-36E	1	AdcDemux.ConvertCode[0] = \$000000;	AdcDemux.ConvertCode[0] = \$800800;
	2	AdcDemux.ConvertCode[1] = \$100100;	AdcDemux.ConvertCode[1] = \$900900;
	3	AdcDemux.ConvertCode[2] = \$200200;	AdcDemux.ConvertCode[2] = \$A00A00;
	4	AdcDemux.ConvertCode[3] = \$300300;	AdcDemux.ConvertCode[3] = \$B00B00;
	5	AdcDemux.ConvertCode[4] = \$400400;	AdcDemux.ConvertCode[4] = \$C00C00;
	6	AdcDemux.ConvertCode[5] = \$500500;	AdcDemux.ConvertCode[5] = \$D00D00;
	7	AdcDemux.ConvertCode[6] = \$600600;	AdcDemux.ConvertCode[6] = \$E00E00;
	8	AdcDemux.ConvertCode[7] = \$700700;	AdcDemux.ConvertCode[7] = \$F00F00;
2 <sup>nd</sup> ACC-36E	9	AdcDemux.ConvertCode[8] = \$000000;	AdcDemux.ConvertCode[8] = \$800800;
	10	AdcDemux.ConvertCode[9] = \$100100;	AdcDemux.ConvertCode[9] = \$900900;
	11	AdcDemux.ConvertCode[10] = \$200200;	AdcDemux.ConvertCode[10] = \$A00A00;
	12	AdcDemux.ConvertCode[11] = \$300300;	AdcDemux.ConvertCode[11] = \$B00B00;
	13	AdcDemux.ConvertCode[12] = \$400400;	AdcDemux.ConvertCode[12] = \$C00C00;
	14	AdcDemux.ConvertCode[13] = \$500500;	AdcDemux.ConvertCode[13] = \$D00D00;
	15	AdcDemux.ConvertCode[14] = \$600600;	AdcDemux.ConvertCode[14] = \$E00E00;
	16	AdcDemux.ConvertCode[15] = \$700700;	AdcDemux.ConvertCode[15] = \$F00F00;

### 4. Accessing the A/D Registers:

The resulting data is found in `AdcDemux.ResultLow[i]` and `AdcDemux.ResultHigh[i]`.

### Accessing AdcDemux Structures in C Code (Optional; For C Programmers)

Having configured the Power PMAC to process the ADC inputs, the following example shows how to access the data in C environment. It reads the 16-channel ADCs in a CPLC and stores them in a user defined array (**ADC\_Result[]** in the following example) whose elements can subsequently be used from within the C program as the user desires.

```
#include <RtGpShm.h>
#include <stdio.h>
#include <dlfcn.h>

void user_plcc()
{
    /***** ADC Result Array Format *****/
    ADC_Result[0] stores ADC Channel 1 result
    ADC_Result[1] stores ADC Channel 2 result
    ADC_Result[2] stores ADC Channel 3 result
    ADC_Result[3] stores ADC Channel 4 result
    ADC_Result[4] stores ADC Channel 5 result
    ADC_Result[5] stores ADC Channel 6 result
    ADC_Result[6] stores ADC Channel 7 result
    ADC_Result[7] stores ADC Channel 8 result
    ADC_Result[8] stores ADC Channel 9 result
    ADC_Result[9] stores ADC Channel 10 result
    ADC_Result[10] stores ADC Channel 11 result
    ADC_Result[11] stores ADC Channel 12 result
    ADC_Result[12] stores ADC Channel 13 result
    ADC_Result[13] stores ADC Channel 14 result
    ADC_Result[14] stores ADC Channel 15 result
    ADC_Result[15] stores ADC Channel 16 result*/
    unsigned int index;
    // ADC Result Storage Array
    int ADC_Result[16]; // int assumes bipolar signal; use unsigned int if unipolar
    // Access ADC results one by one in a for loop
    for(index = 0; index < 16; index++)
    {
        // Store ADC Results in array elements
        ADC_Result[index] = pshm->AdcDemux.ResultLow[index];
        ADC_Result[index + 8] = pshm->AdcDemux.ResultHigh[index];
    }
    return;
}
```

## Using an Analog Input for Servo Feedback

The ACC-36E analog inputs can be used as a feedback device for a servo motor. For simplicity, the automatic ADC read function is recommended for this application. This example assumes that the automatic ADC read function has already been configured and activated.

**Example:** Setting up Motor #1 with position and velocity feedback from ADC channel 1. The analog input is brought into the Encoder Conversion Table (ECT) as a single read of a 32-bit register (in the Power PMAC IDE Software: Delta Tau → Configure → Encoder Conversion Table):

Number	Type	pEnc	pEnc1	MaxDelta
1	1	AdcDemux.ResultLow[0].a	Sys.pushm	0

The equivalent Power PMAC Script code:

```
EncTable[1].type = 1; // Set entry type to 32-bit word read
EncTable[1].pEnc = AdcDemux.ResultLow[0].a; // Set encoder address to ADC channel 1
EncTable[1].pEnc1 = Sys.pushm; // Unused; set to Sys.pushm
EncTable[1].index1 = 20; // Shift left 20 to put source MSB in bit 31
EncTable[1].index2 = 0; // Shift left 0 to put source LSB into bit 0
EncTable[1].index3 = 0; // No limit on first derivative
EncTable[1].index4 = 0; // No limit on second derivative
EncTable[1].ScaleFactor = 1/pow(2,EncTable[1].index1); // Scale factor (1/(2^20))
```

The position and velocity pointers are then set to the processed data address:

```
Motor[1].pEnc = EncTable[1].a; // Outer (position) loop source address
Motor[1].pEnc2 = EncTable[1].a; // Inner (velocity) loop source address
```

## Analog Input Power-On Position

Some analog devices are absolute along the travel range of the motor (e.g., in hydraulic piston applications). The following example code will configure Motor #1 to use ADC channel 1 for the power-on position read as unsigned data with a scale factor of 1:

```
Motor[1].pAbsPos = AdcDemux.ResultLow[0].a; // Set position register to ADC channel 1
Motor[1].AbsPosFormat = $00000C00;          // Use 12 bits starting at bit 0, unsigned
                                              // (for signed, change to $01000C00)
Motor[1].AbsPosSf = 1;                       // Scale factor of 1
```

## Manual ADC Read Using ACC-36E Structures

Power PMAC supports the following structures for manually reading ACC-36E:

Structure	Description
<b>ACC36E[n].ConvertCode</b>	ADC pair and convert code select
<b>ACC36E[n].ADCRdyLow</b>	Ready-to-read bit for ADC#1 through #8
<b>ACC36E[n].ADCRdyHigh</b>	Ready-to-read bit for ADC#9 through #16
<b>ACC36E[n].ADCuLow</b>	Read ADC#1 through #8 as unipolar
<b>ACC36E[n].ADCuHigh</b>	Read ADC#9 through #16 as unipolar
<b>ACC36E[n].ADCsLow</b>	Read ADC#1 through #8 as bipolar
<b>ACC36E[n].ADCsHigh</b>	Read ADC#9 through #16 as bipolar
<b>ACC36E[n].ADCHighLow</b>	Read ADC pair as one word



**Note**

$n$  is the index specified by the DIP switch SW1 setting of the ACC-36E.

Setting up the manual ADC read with Power PMAC requires two simple steps:

1. Configure **ACC36E[n].ConvertCode** = (ADC Pair #) – 1 for Unipolar Inputs  
= (ADC Pair #) + 7 for Bipolar Inputs

ADC Pair #	ADC#	Channel Select Pointer Value	
		Unipolar Inputs	Bipolar Inputs
1	1 & 9	0	8
2	2 & 10	1	9
3	3 & 11	2	10
4	4 & 12	3	11
5	5 & 13	4	12
6	6 & 14	5	13
7	7 & 15	6	14
8	8 & 16	7	15

2. Read/copy ADC data results as soon as the **ADC Ready Bits** are “ready” (that is, their values become 1). This can be achieved easily in a Power PMAC script PLC, and implemented for all channels to be read consecutively and consistently, creating a “custom automatic” ADC read function.

## ADC Manual Read Example Script PLCs

Setting up a Power UMAC, with an ACC-36E at I/O base address offset \$A00000, to read channels 1 through 16 and store the results in global variables ADC1 through ADC16:

### Unipolar Example

```

global ADC1;           // Channel 1 ADC storage variable
global ADC2;           // Channel 2 ADC storage variable
global ADC3;           // Channel 3 ADC storage variable
global ADC4;           // Channel 4 ADC storage variable
global ADC5;           // Channel 5 ADC storage variable
global ADC6;           // Channel 6 ADC storage variable
global ADC7;           // Channel 7 ADC storage variable
global ADC8;           // Channel 8 ADC storage variable
global ADC9;           // Channel 9 ADC storage variable
global ADC10;          // Channel 10 ADC storage variable
global ADC11;          // Channel 11 ADC storage variable
global ADC12;          // Channel 12 ADC storage variable
global ADC13;          // Channel 13 ADC storage variable
global ADC14;          // Channel 14 ADC storage variable
global ADC15;          // Channel 15 ADC storage variable
global ADC16;          // Channel 16 ADC storage variable

Open PLC 1
ACC36E[0].ConvertCode = 0;           // Select Channel 1 & 9, unipolar
While (ACC36E[0].ADCRdyLow != 1 && ACC36E[0].ADCRdyHigh != 1){} // Wait for ADCs
ADC1 = ACC36E[0].ADCuLow;            // Read and copy result into storage
ADC9 = ACC36E[0].ADCuHigh;           // Read and copy result into storage

ACC36E[0].ConvertCode = 1;           // Select Channel 2 & 10, unipolar
While (ACC36E[0].ADCRdyLow != 1 && ACC36E[0].ADCRdyHigh != 1){} // Wait for ADCs
ADC2 = ACC36E[0].ADCuLow;            // Read and copy result into storage
ADC10 = ACC36E[0].ADCuHigh;          // Read and copy result into storage

ACC36E[0].ConvertCode = 2;           // Select Channel 3 & 11, unipolar
While (ACC36E[0].ADCRdyLow != 1 && ACC36E[0].ADCRdyHigh != 1){} // Wait for ADCs
ADC3 = ACC36E[0].ADCuLow;            // Read and copy result into storage
ADC11 = ACC36E[0].ADCuHigh;          // Read and copy result into storage

ACC36E[0].ConvertCode = 3;           // Select Channel 4 & 12, unipolar
While (ACC36E[0].ADCRdyLow != 1 && ACC36E[0].ADCRdyHigh != 1){} // Wait for ADCs
ADC4 = ACC36E[0].ADCuLow;            // Read and copy result into storage
ADC12 = ACC36E[0].ADCuHigh;          // Read and copy result into storage

ACC36E[0].ConvertCode = 4;           // Select Channel 5 & 13, unipolar
While (ACC36E[0].ADCRdyLow != 1 && ACC36E[0].ADCRdyHigh != 1){} // Wait for ADCs
ADC5 = ACC36E[0].ADCuLow;            // Read and copy result into storage
ADC13 = ACC36E[0].ADCuHigh;          // Read and copy result into storage

ACC36E[0].ConvertCode = 5;           // Select Channel 6 & 14, unipolar
While (ACC36E[0].ADCRdyLow != 1 && ACC36E[0].ADCRdyHigh != 1){} // Wait for ADCs
ADC6 = ACC36E[0].ADCuLow;            // Read and copy result into storage
ADC14 = ACC36E[0].ADCuHigh;          // Read and copy result into storage

ACC36E[0].ConvertCode = 6;           // Select Channel 7 & 15, unipolar
While (ACC36E[0].ADCRdyLow != 1 && ACC36E[0].ADCRdyHigh != 1){} // Wait for ADCs
ADC7 = ACC36E[0].ADCuLow;            // Read and copy result into storage
ADC15 = ACC36E[0].ADCuHigh;          // Read and copy result into storage

ACC36E[0].ConvertCode = 7;           // Select Channel 8 & 16, unipolar
While (ACC36E[0].ADCRdyLow != 1 && ACC36E[0].ADCRdyHigh != 1){} // Wait for ADCs
ADC8 = ACC36E[0].ADCuLow;            // Read and copy result into storage
ADC16 = ACC36E[0].ADCuHigh;          // Read and copy result into storage
Close

```



**Bipolar Example**

```

global ADC1;           // Channel 1 ADC storage variable
global ADC2;           // Channel 2 ADC storage variable
global ADC3;           // Channel 3 ADC storage variable
global ADC4;           // Channel 4 ADC storage variable
global ADC5;           // Channel 5 ADC storage variable
global ADC6;           // Channel 6 ADC storage variable
global ADC7;           // Channel 7 ADC storage variable
global ADC8;           // Channel 8 ADC storage variable
global ADC9;           // Channel 9 ADC storage variable
global ADC10;          // Channel 10 ADC storage variable
global ADC11;          // Channel 11 ADC storage variable
global ADC12;          // Channel 12 ADC storage variable
global ADC13;          // Channel 13 ADC storage variable
global ADC14;          // Channel 14 ADC storage variable
global ADC15;          // Channel 15 ADC storage variable
global ADC16;          // Channel 16 ADC storage variable

Open PLC 1
ACC36E[0].ConvertCode = 8;           // Select Channel 1 & 9, bipolar
While (ACC36E[0].ADCRdyLow != 1 && ACC36E[0].ADCRdyHigh != 1){} // Wait for ADCs
ADC1 = ACC36E[0].ADCsLow;             // Read and copy result into storage
ADC9 = ACC36E[0].ADCsHigh;            // Read and copy result into storage

ACC36E[0].ConvertCode = 9;           // Select Channel 2 & 10, bipolar
While (ACC36E[0].ADCRdyLow != 1 && ACC36E[0].ADCRdyHigh != 1){} // Wait for ADCs
ADC2 = ACC36E[0].ADCsLow;             // Read and copy result into storage
ADC10 = ACC36E[0].ADCsHigh;           // Read and copy result into storage

ACC36E[0].ConvertCode = 10;          // Select Channel 3 & 11, bipolar
While (ACC36E[0].ADCRdyLow != 1 && ACC36E[0].ADCRdyHigh != 1){} // Wait for ADCs
ADC3 = ACC36E[0].ADCsLow;             // Read and copy result into storage
ADC11 = ACC36E[0].ADCsHigh;           // Read and copy result into storage

ACC36E[0].ConvertCode = 11;          // Select Channel 4 & 12, bipolar
While (ACC36E[0].ADCRdyLow != 1 && ACC36E[0].ADCRdyHigh != 1){} // Wait for ADCs
ADC4 = ACC36E[0].ADCsLow;             // Read and copy result into storage
ADC12 = ACC36E[0].ADCsHigh;           // Read and copy result into storage

ACC36E[0].ConvertCode = 12;          // Select Channel 5 & 13, bipolar
While (ACC36E[0].ADCRdyLow != 1 && ACC36E[0].ADCRdyHigh != 1){} // Wait for ADCs
ADC5 = ACC36E[0].ADCsLow;             // Read and copy result into storage
ADC13 = ACC36E[0].ADCsHigh;           // Read and copy result into storage

ACC36E[0].ConvertCode = 13;          // Select Channel 6 & 14, bipolar
While (ACC36E[0].ADCRdyLow != 1 && ACC36E[0].ADCRdyHigh != 1){} // Wait for ADCs
ADC6 = ACC36E[0].ADCsLow;             // Read and copy result into storage
ADC14 = ACC36E[0].ADCsHigh;           // Read and copy result into storage

ACC36E[0].ConvertCode = 14;          // Select Channel 7 & 15, bipolar
While (ACC36E[0].ADCRdyLow != 1 && ACC36E[0].ADCRdyHigh != 1){} // Wait for ADCs
ADC7 = ACC36E[0].ADCsLow;             // Read and copy result into storage
ADC15 = ACC36E[0].ADCsHigh;           // Read and copy result into storage

ACC36E[0].ConvertCode = 15;          // Select Channel 8 & 16, bipolar
While (ACC36E[0].ADCRdyLow != 1 && ACC36E[0].ADCRdyHigh != 1){} // Wait for ADCs
ADC8 = ACC36E[0].ADCsLow;             // Read and copy result into storage
ADC16 = ACC36E[0].ADCsHigh;           // Read and copy result into storage
Close

```

## Accessing ADCs from C Environment (For C Programmers)

### Reading:

The **GetPmacVar()** function below allows one to read the ACC-36E structures in C. It is defined as **int GetPmacVar (char \*pinstr, double \*pdata)**, where:

- The first argument is a string containing the name of the variable to query.
- The second argument is a double containing the address of the ADC result (low or high).

### Writing:

The **SetPmacVar()** function below allows writing to the ACC-36E structures in C. It is defined as: **int SetPmacVar (char \*pinstr, double data)**, where:

- The first argument is a string containing the name of the variable to which to write.
- The second argument is a double containing the value to write.



When using C routines, failure to release control of the loop waiting for the ADC conversions to finish may cause PMAC to lock up, possibly creating a runaway condition. Thus, one of the following functions, **WaitForADC()**, has precautions in order to release control of the wait loop in the event that the ADC conversion bits never become 1.

---

### Example: Unipolar Manual ADC Read in a CPLC

Configuring one ACC-36E with unipolar inputs at base offset \$A00000.

```
#include <gplib.h>
#include <stdio.h>
#include <dlfcn.h>

// Definition(s)
#define FirstCardNumber      0          // At Base Offset $A00000

// Prototype(s)
int WaitForADC(unsigned int Card_Index);

void user_plcc() {
    unsigned int ADC_Result_Unipolar[16],index;
    int waitResult = 0;
    char buffer[32]="";
    double temp = 0,ConvertCode = 0;
    // Access ADC results one by one in a for loop
    for(index = 0; index < 8; index++) {
        // Prepare string
        sprintf(buffer,"ACC36E[%d].ConvertCode",FirstCardNumber);
        ConvertCode = index;          // Compute convert code
        // Write Unipolar convert code to Channel Select structure
        SetPmacVar(buffer,ConvertCode);
        // Poll conversion complete bits until the conversions are finished
        waitResult = WaitForADC(FirstCardNumber);
        if(waitResult < 0)
        {
            return;                  // Conversion was not successful
        }
        // Prepare string
        sprintf(buffer,"ACC36E[%d].ADCuLow",FirstCardNumber);
        GetPmacVar(buffer,&temp);      // Read the structure
        // Cast and store low unipolar (unsigned) ADC Result in array element
        ADC_Result_Unipolar[index] = (unsigned int)temp;
    }
}
```

```

        // Prepare string
        sprintf(buffer,"ACC36E[%d].ADCuHigh",FirstCardNumber);
        GetPmacVar(buffer,&temp);        // Read the structure
        // Cast and store high unipolar (unsigned) ADC Result in array element
        ADC_Result_Unipolar[index + 8] = (unsigned int)temp;
        /****Insert user code here, where ADC_Result_Unipolar elements
        and ADC_Result_Bipolar elements
        can be used for the user's calculations*****/
        // For debugging purposes:
        pshm->P[1000 + index] = ADC_Result_Unipolar[index];
        pshm->P[1000 + index + 8] = ADC_Result_Unipolar[index + 8];
    }
    return;
}

int WaitForADC(unsigned int Card_Index)
{
    // Waits until ADC conversions have completed
    // Inputs:
    // Card_Index: index (n) from POWER section of Addressing ACC-36E table

    // Outputs:
    // returns 0 if successfully performed ADC conversion
    // returns -1 if conversion did not complete within Timeout ms
    unsigned int RdyLow = 0,RdyHigh = 0,iterations = 0;
    double Present_Time,Conversion_Start_Time,Time_Difference,Timeout,Timeout_us,temp = 0;
    char str1[24]="",str2[24]="";
    struct timespec SleepTime={0};
    SleepTime.tv_nsec=1000000;
    sprintf(str1,"ACC36E[%u].ADCRdyLow",Card_Index);    // Prepare string
    sprintf(str2,"ACC36E[%u].ADCRdyHigh",Card_Index);    // Prepare string
    // Get time at (almost) start of conversion (microseconds)
    Conversion_Start_Time = GetCPUClock();
    // Timeout: Maximum permitted time to wait for ADC conversion to finish before error
    // (milliseconds)
    Timeout = 500; // Milliseconds
    Timeout_us = Timeout*1000;    // Convert to microseconds
    do
    {
        // If the loop has taken a multiple of 50 iterations to finish
        if(iterations == 50)
        {
            // Release control for 1 ms so PMAC does not go into Watchdog mode while
            // waiting for conversion to finish
            nanosleep(&SleepTime,NULL); // Release thread and wait 1 msec
            iterations = 0;    // Reset iteration counter
        }
        Present_Time = GetCPUClock(); // Obtain current system time
        // Compute difference in time between starting conversion and now
        Time_Difference = Present_Time-Conversion_Start_Time;
        if(Time_Difference > Timeout_us)    // If more than Timeout ms have elapsed
        {
            return (-1);    // Return with error code
        }
        GetPmacVar(str1,&temp);    // Read the ADCRdyLow structure
        RdyLow = (unsigned int)temp;
        GetPmacVar(str2,&temp);    // Read the ADCRdyHigh structure
        RdyHigh = (unsigned int)temp;
        iterations++;
    } while(RdyLow != 1 && RdyHigh != 1); // Test ADC ready bit
    return 0;    // Return with success code
}

```

**Example: Bipolar Manual ADC Read in a CPLC**

Configuring one ACC-36E with bipolar inputs at base offset \$A00000.

```
#include <gplib.h>
#include <stdio.h>
#include <dlfcn.h>

// Definition(s)
#define FirstCardNumber      0          // At Base Offset $A00000

// Prototype(s)
int WaitForADC(unsigned int Card_Index);

void user_plcc() {
    unsigned int index;
    int ADC_Result_Bipolar[16],waitResult = 0;
    char buffer[32]="";
    double temp = 0,ConvertCode = 0;
    // Access ADC results one by one in a for loop
    for(index = 0; index < 8; index++) {
        // Prepare string
        sprintf(buffer,"ACC36E[%d].ConvertCode",FirstCardNumber);
        ConvertCode = index + 8;          // Compute convert code
        // Write bipolar convert code to Channel Select structure
        SetPmacVar(buffer,ConvertCode);
        // Poll conversion complete bits until the conversions are finished
        waitResult = WaitForADC(FirstCardNumber);
        if(waitResult < 0)
        {
            return;                      // Conversion was not successful
        }
        // Prepare string
        sprintf(buffer,"ACC36E[%d].ADCsLow",FirstCardNumber);
        GetPmacVar(buffer,&temp);          // Read the structure
        // Cast and store low bipolar (signed) ADC Result in array element
        ADC_Result_Bipolar[index] = (int)temp;
        // Prepare string
        sprintf(buffer,"ACC36E[%d].ADCsHigh",FirstCardNumber);
        GetPmacVar(buffer,&temp);          // Read the structure
        // Cast and store high bipolar (signed) ADC Result in array element
        ADC_Result_Bipolar[index + 8] = (int)temp;
        /*****Insert user code here, where ADC_Result_Unipolar elements
        and ADC_Result_Bipolar elements
        can be used for the user's calculations*****/
        // For debugging purposes:
        pshm->P[1000 + index] = ADC_Result_Bipolar[index];
        pshm->P[1000 + index + 8] = ADC_Result_Bipolar[index + 8];
    }
    return;
}

int WaitForADC(unsigned int Card_Index)
{
    // Waits until ADC conversions have completed
    // Inputs:
    // Card_Index: index (n) from POWER section of Addressing ACC-36E table

    // Outputs:
    // returns 0 if successfully performed ADC conversion
    // returns -1 if conversion did not complete within Timeout ms
    unsigned int RdyLow = 0,RdyHigh = 0,iterations = 0;
    double Present_Time,Conversion_Start_Time,Time_Difference,Timeout,Timeout_us,temp = 0;
    char str1[24]="",str2[24]="";
    struct timespec SleepTime={0};
    SleepTime.tv_nsec=1000000;
    sprintf(str1,"ACC36E[%u].ADCRdyLow",Card_Index);    // Prepare string
    sprintf(str2,"ACC36E[%u].ADCRdyHigh",Card_Index);  // Prepare string
    // Get time at (almost) start of conversion (microseconds)
    Conversion_Start_Time = GetCPUClock();

    // Timeout: Maximum permitted time to wait for ADC conversion to finish before error
```

```
// (milliseconds)
Timeout = 500; // Milliseconds
Timeout_us = Timeout*1000; // Convert to microseconds
do
{
    // If the loop has taken a multiple of 50 iterations to finish
    if(iterations == 50)
    {
        // Release control for 1 ms so PMAC does not go into Watchdog mode while
        // waiting for conversion to finish
        nanosleep(&SleepTime,NULL); // Release thread and wait 1 msec
        iterations = 0; // Reset iteration counter
    }
    Present_Time = GetCPUClock(); // Obtain current system time
    // Compute difference in time between starting conversion and now
    Time_Difference = Present_Time-Conversion_Start_Time;
    if(Time_Difference > Timeout_us) // If more than Timeout ms have elapsed
    {
        return (-1); // Return with error code
    }
    GetPmacVar(str1,&temp); // Read the ADCRdyLow structure
    RdyLow = (unsigned int)temp;
    GetPmacVar(str2,&temp); // Read the ADCRdyHigh structure
    RdyHigh = (unsigned int)temp;
    iterations++;
} while(RdyLow != 1 && RdyHigh != 1); // Test ADC ready bit
return 0; // Return with success code
}
```

**Example: Unipolar/Bipolar ADC Read in a CPLC**

Configuring two ACC-36E cards: 1<sup>st</sup> card has unipolar inputs and is at base offset \$A00000, and 2<sup>nd</sup> card has bipolar inputs at base offset \$B00000.

```
#include <gplib.h>
#include <stdio.h>
#include <dlfcn.h>

// Definition(s)
#define FirstCardNumber    0        // At Base Offset $A00000
#define SecondCardNumber  1        // At Base Offset $B00000

// Prototype(s)
int WaitForADC(unsigned int Card_Index);

void user_plcc() {
    unsigned int ADC_Result_Unipolar[16],index;
    int ADC_Result_Bipolar[16],waitResult = 0;
    char buffer[32]="";
    double temp = 0,ConvertCode = 0;
    // Access ADC results one by one in a for loop
    for(index = 0; index < 8; index++) {
        // Prepare string
        sprintf(buffer,"ACC36E[%d].ConvertCode",FirstCardNumber);
        ConvertCode = index;          // Compute convert code
        // Write Unipolar convert code to Channel Select structure
        SetPmacVar(buffer,ConvertCode);
        // Poll conversion complete bits until the conversions are finished
        waitResult = WaitForADC(FirstCardNumber);
        if(waitResult < 0)
        {
            return;                  // Conversion was not successful
        }
        // Prepare string
        sprintf(buffer,"ACC36E[%d].ADCuLow",FirstCardNumber);
        GetPmacVar(buffer,&temp);      // Read the structure
        // Cast and store low unipolar (unsigned) ADC Result in array element
        ADC_Result_Unipolar[index] = (unsigned int)temp;
        // Prepare string
        sprintf(buffer,"ACC36E[%d].ADCuHigh",FirstCardNumber);
        GetPmacVar(buffer,&temp);      // Read the structure
        // Cast and store high unipolar (unsigned) ADC Result in array element
        ADC_Result_Unipolar[index + 8] = (unsigned int)temp;
        // Prepare string
        sprintf(buffer,"ACC36E[%d].ConvertCode",SecondCardNumber);
        ConvertCode = index + 8;      // Compute convert code
        // Write bipolar convert code to Channel Select structure
        SetPmacVar(buffer,ConvertCode);
        // Poll conversion complete bits until the conversions are finished
        waitResult = WaitForADC(SecondCardNumber);
        if(waitResult < 0)
        {
            return;                  // Conversion was not successful
        }
        // Prepare string
        sprintf(buffer,"ACC36E[%d].ADCsLow",SecondCardNumber);
        GetPmacVar(buffer,&temp);      // Read the structure
        // Cast and store low bipolar (signed) ADC Result in array element
        ADC_Result_Bipolar[index] = (int)temp;
        // Prepare string
        sprintf(buffer,"ACC36E[%d].ADCsHigh",SecondCardNumber);
        GetPmacVar(buffer,&temp);      // Read the structure
        // Cast and store high bipolar (signed) ADC Result in array element
        ADC_Result_Bipolar[index + 8] = (int)temp;
        /*****Insert user code here, where ADC_Result_Unipolar elements
        and ADC_Result_Bipolar elements
        can be used for the user's calculations*****/
    }
    return;
}

int WaitForADC(unsigned int Card_Index)
```

```

{
    // Waits until ADC conversions have completed
    // Inputs:
    // Card_Index: index (n) from POWER section of Addressing ACC-36E table

    // Outputs:
    // returns 0 if successfully performed ADC conversion
    // returns -1 if conversion did not complete within Timeout ms
    unsigned int RdyLow = 0, RdyHigh = 0, iterations = 0;
    double Present_Time, Conversion_Start_Time, Time_Difference, Timeout, Timeout_us, temp = 0;
    char str1[24]="", str2[24]="";
    struct timespec SleepTime={0};
    SleepTime.tv_nsec=1000000;
    sprintf(str1,"ACC36E[%u].ADCRdyLow",Card_Index);    // Prepare string
    sprintf(str2,"ACC36E[%u].ADCRdyHigh",Card_Index);  // Prepare string
    // Get time at (almost) start of conversion (microseconds)
    Conversion_Start_Time = GetCPUClock();
    // Timeout: Maximum permitted time to wait for ADC conversion to finish before error
    // (milliseconds)
    Timeout = 500; // Milliseconds
    Timeout_us = Timeout*1000;    // Convert to microseconds
    do
    {
        // If the loop has taken a multiple of 50 iterations to finish
        if(iterations == 50)
        {
            // Release control for 1 ms so PMAC does not go into Watchdog mode while
            // waiting for conversion to finish
            nanosleep(&SleepTime,NULL); // Release thread and wait 1 msec
            iterations = 0;    // Reset iteration counter
        }
        Present_Time = GetCPUClock(); // Obtain current system time
        // Compute difference in time between starting conversion and now
        Time_Difference = Present_Time-Conversion_Start_Time;
        if(Time_Difference > Timeout_us)    // If more than Timeout ms have elapsed
        {
            return (-1);    // Return with error code
        }
        GetPmacVar(str1,&temp);    // Read the ADCRdyLow structure
        RdyLow = (unsigned int)temp;
        GetPmacVar(str2,&temp);    // Read the ADCRdyHigh structure
        RdyHigh = (unsigned int)temp;
        iterations++;
    } while(RdyLow != 1 && RdyHigh != 1); // Test ADC ready bit
    return 0;    // Return with success code
}

```

## Testing the Analog Inputs

The Analog Inputs can be brought into the ACC-36E as single ended (ADC+ & Ground) or differential (ADC+ & ADC-) signals.



### *Note*

In single-ended mode, ADC- should to be tied to analog ground for full resolution and proper operation.

Reading the input signals in software counts using the predefined M-Variables should show the following:

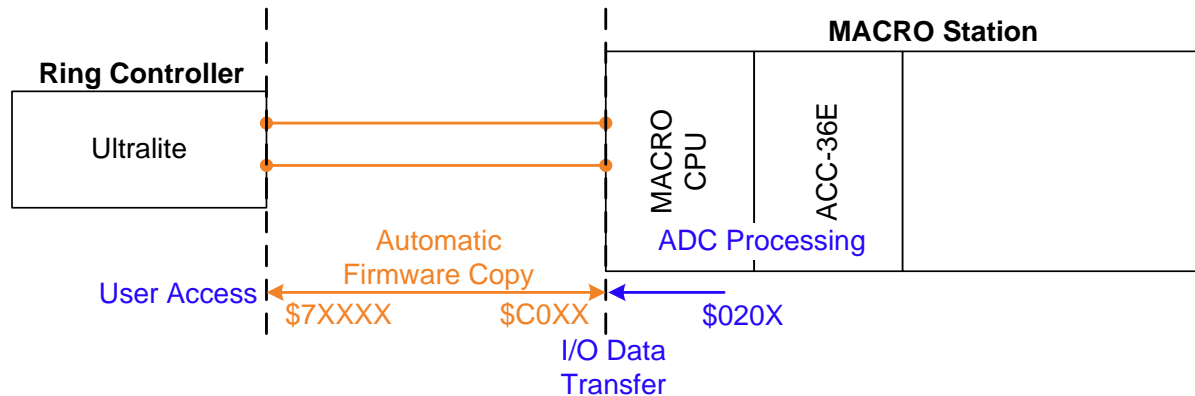
ADC Input	Single-Ended [V] ADC+ <=> AGND	Differential [V] ADC+ <=> ADC-	Software Counts
Unipolar Mode	0	0	0
	10	10	2047
	20	20	4095
Bipolar Mode	-10	-10	-2047
	0	0	0
	10	10	2047



## USING ACC-36E WITH UMAC MACRO

Setting up the ACC-36E on a MACRO station requires the following steps:

- Establishing communication with the MACRO Station and enabling nodes
- Enabling ADC Processing (Automatic Read Function) at the MACRO Station
- Transferring Data Over I/O Nodes



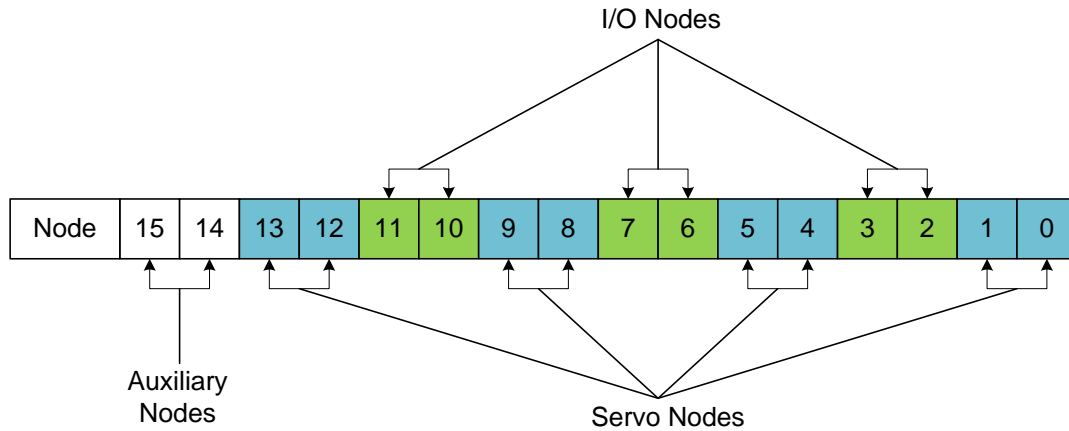
The goal is to allow the user “software” access to the analog inputs brought into the MACRO Station(s) from the Ring Controller (i.e. Turbo PMAC2 Ultralite, or UMAC with ACC-5E).

The automatic read function (ADC processing) demultiplexes the ADC data and puts the ADC data into predefined “local” registers (\$02XX) at the MACRO Station side. The I/O node data transfer then copies the data from these registers (\$02XX) into MACRO Station node registers (\$C0XX), which are in turn automatically copied by the firmware (with no additional user settings required) into their complement node registers (\$78XXX) at the Ring Controller side.

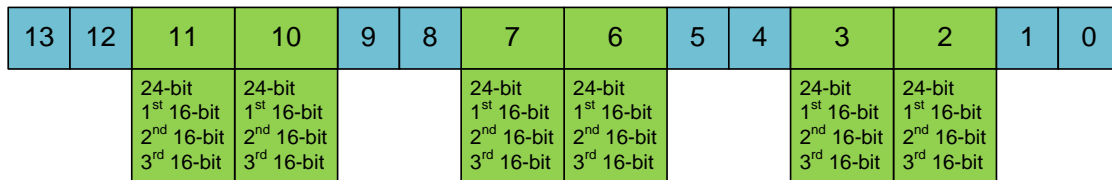
## Quick Review: Nodes and Addressing

Each MACRO IC consists of 16 nodes: 2 auxiliary, 8 servo, and 6 I/O nodes.

- Auxiliary nodes are Master/Control registers and internal firmware use.
- Servo nodes are used for motor control, carrying feedback, commands, and flag information.
- I/O nodes are by default unoccupied and are user configurable for transferring various data.



Each I/O node consists of 4 registers; one 24-bit and three 16-bit registers for a total of 72 bits of data:



A given MACRO Station can be populated with either a MACRO8 or MACRO16 CPU:

- MACRO8 supports only 1 MACRO IC (IC#0).
- MACRO16 supports 2 MACRO ICs (IC#0 and IC#1).

The I/O node addresses (\$C0XX) for each of the Station MACRO ICs are:

Station MACRO IC #0 Node Registers						
Node	2	3	6	7	10	11
24-bit	X:\$C0A0	X:\$C0A4	X:\$C0A8	X:\$C0AC	X:\$C0B0	X:\$C0B4
16-bit	X:\$C0A1	X:\$C0A5	X:\$C0A9	X:\$C0AD	X:\$C0B1	X:\$C0B5
16-bit	X:\$C0A2	X:\$C0A6	X:\$C0AA	X:\$C0AE	X:\$C0B2	X:\$C0B6
16-bit	X:\$C0A3	X:\$C0A7	X:\$C0AB	X:\$C0AF	X:\$C0B3	X:\$C0B7

Station MACRO IC #1 Node Registers						
Node	2	3	6	7	10	11
24-bit	X:\$C0E0	X:\$C0E4	X:\$C0E8	X:\$C0EC	X:\$C0F0	X:\$C0F4
16-bit	X:\$C0E1	X:\$C0E5	X:\$C0E9	X:\$C0ED	X:\$C0F1	X:\$C0F5
16-bit	X:\$C0E2	X:\$C0E6	X:\$C0EA	X:\$C0EE	X:\$C0F2	X:\$C0F6
16-bit	X:\$C0E3	X:\$C0E7	X:\$C0EB	X:\$C0EF	X:\$C0F3	X:\$C0F7



Non-Turbo PMAC2 Ultralite (legacy) I/O node addresses are the same as Station MACRO IC#0 node registers.

**Note**

A given Turbo PMAC2 Ultralite (or UMAC with ACC-5E) can be populated with up to 4 MACRO ICs (IC#0, IC#1, IC#2, and IC#3) which can be queried with global variable I4902:

If I4902=	Populated MACRO IC #s
\$0	None
\$1	0
\$3	0, 1
\$7	0, 1, 2
\$F	0, 1, 2, 3

And the I/O node addresses (\$7XXXX) for each of the Ultralite MACRO ICs are:

Ring Controller MACRO IC #0 Node Registers						
Station I/O Node#	2	3	6	7	10	11
Ultralite I/O Node#	2	3	6	7	10	11
24-bit	X:\$78420	X:\$78424	X:\$78428	X:\$7842C	X:\$78430	X:\$78434
16-bit	X:\$78421	X:\$78425	X:\$78429	X:\$7842D	X:\$78431	X:\$78435
16-bit	X:\$78422	X:\$78426	X:\$7842A	X:\$7842E	X:\$78432	X:\$78436
16-bit	X:\$78423	X:\$78427	X:\$7842B	X:\$7842F	X:\$78433	X:\$78437

Ring Controller MACRO IC #1 Node Registers						
Station I/O Node#	2	3	6	7	10	11
Ultralite I/O Node#	18	19	22	23	26	27
24-bit	X:\$79420	X:\$79424	X:\$79428	X:\$7942C	X:\$79430	X:\$79434
16-bit	X:\$79421	X:\$79425	X:\$79429	X:\$7942D	X:\$79431	X:\$79435
16-bit	X:\$79422	X:\$79426	X:\$7942A	X:\$7942E	X:\$79432	X:\$79436
16-bit	X:\$79423	X:\$79427	X:\$7942B	X:\$7942F	X:\$79433	X:\$79437

Ring Controller MACRO IC #2 Node Registers						
Station I/O Node#	2	3	6	7	10	11
Ultralite I/O Node#	34	35	38	39	42	43
24-bit	X:\$7A420	X:\$7A424	X:\$7A428	X:\$7A42C	X:\$7A430	X:\$7A434
16-bit	X:\$7A421	X:\$7A425	X:\$7A429	X:\$7A42D	X:\$7A431	X:\$7A435
16-bit	X:\$7A422	X:\$7A426	X:\$7A42A	X:\$7A42E	X:\$7A432	X:\$7A436
16-bit	X:\$7A423	X:\$7A427	X:\$7A42B	X:\$7A42F	X:\$7A433	X:\$7A437

Ring Controller MACRO IC #3 Node Registers						
Station I/O Node#	2	3	6	7	10	11
Ultralite I/O Node#	50	51	54	55	58	59
24-bit	X:\$7B420	X:\$7B424	X:\$7B428	X:\$7B42C	X:\$7B430	X:\$7B434
16-bit	X:\$7B421	X:\$7B425	X:\$7B429	X:\$7B42D	X:\$7B431	X:\$7B435
16-bit	X:\$7B422	X:\$7B426	X:\$7B42A	X:\$7B42E	X:\$7B432	X:\$7B436
16-bit	X:\$7B423	X:\$7B427	X:\$7B42B	X:\$7B42F	X:\$7B433	X:\$7B437

## Enabling MACRO Station ADC Processing

The A/D converter chips used on the ACC-36E multiplex the resulting data, and therefore it is mandatory to read each input one at a time. The only practical way to do this on a MACRO station is using the automatic “built-in” method. This method copies the ADC inputs from the ACC-36E to predefined MACRO station memory locations.



**Note**

This section assumes the necessary I/O nodes have been activated.  
The automatic ADC copy feature is available with:  
MACRO8 CPU (602804) firmware version 1.15 or newer  
MACRO16 CPU (603719) firmware version 1.202 or newer

For each MACRO IC, the automatic read function handles 16 ADC channels, to permit one to read a maximum of 32 ADC channels with two MACRO ICs. Hence, with MACRO8 CPUs, the maximum number of possible ADC automatic reads is 16. With MACRO16 CPUs, the maximum number of possible ADC automatic reads is 32. There are 16 ADCs (8 pairs) per base address, i.e., per ACC-36E present, for a maximum of 2 base addresses with 2 ACC-36Es present.

Pair #	1 <sup>st</sup> ACC-36E	Pair #	2 <sup>nd</sup> ACC-36E
1	ADC#1 & ADC#9	9	ADC#1 & ADC#9
2	ADC#2 & ADC#10	10	ADC#2 & ADC#10
3	ADC#3 & ADC#11	11	ADC#3 & ADC#11
4	ADC#4 & ADC#12	12	ADC#4 & ADC#12
5	ADC#5 & ADC#13	13	ADC#5 & ADC#13
6	ADC#6 & ADC#14	14	ADC#6 & ADC#14
7	ADC#7 & ADC#15	15	ADC#7 & ADC#15
8	ADC#8 & ADC#16	16	ADC#8 & ADC#16

Setting up the automatic read function (ADC processing) successfully requires the configuration of the following 3 MACRO Station I-Variables (MI variables):

- MS{anynode},MI987: A/D Input Enable**

MI987 controls whether the ADC processing is enabled or disabled.

It can take one of the following settings: =0 disabled  
=1 enabled

- MS{anynode},MI988: A/D Unipolar/Bipolar Control**

MI988 specifies whether a given pair is setup for bipolar ( $\pm 10$  V) or unipolar (0 to +20 V) mode.

It is an 8-bit word, with each bit strictly controlling a pair of ADCs: Bit value = 0 → Unipolar  
Bit value = 1 → Bipolar

Bit#	Pair
0	ADC1 & ADC9
1	ADC2 & ADC10
2	ADC3 & ADC11
3	ADC4 & ADC12
4	ADC5 & ADC13
5	ADC6 & ADC14
6	ADC7 & ADC15
7	ADC8 & ADC16

For all 16 ADCs unipolar:

MS{ anynode },MI988=\$00

For all 16 ADCs bipolar:

MS{ anynode },MI988=\$FF

- **MS{anynode},MI989: A/D Source Address**

MI989 specifies the card's starting source address for the ADC inputs (dip switch setting).



*Note*

The first 16 ADC transfers use MACRO IC 0 parameters, as described above. The next (and last) 16 ADC transfers use the corresponding MACRO IC#1 parameters:

MS{anynode},MI1989  
MS{anynode},MI1987  
MS{anynode},MI1988

### Automatic ADC processing Examples:

Setting up one ACC-36E (at base address \$8800) to copy 16 ADCs, with ADCs 1, 2, 3, 4, 9, 10, 11, 12 as unipolar and ADCs 5, 6, 7, 8, 13, 14, 15, 16 as bipolar:

```
MS0,MI987=1           ; Enable automatic ADC read function
MS0,MI988=$F0         ; ADCs 1,2,3,4,9,10,11,12 as unipolar
                      ; and ADCs 5,6,7,8,13,14,15,16 as bipolar
MS0,MI989=$8800       ; Card base address
```

Setting up two ACC-36Es (at base addresses \$8800 and \$9800) to copy 32 ADCs, with all 16 ADCs of the 1<sup>st</sup> ACC-36E configured as unipolar and all 16 ADCs of the 2<sup>nd</sup> ACC-36E as bipolar:

```
// Setting up the automatic read function for the 1st ACC-36E
MS0,MI987=1           ; Enable automatic ADC read function (MACRO IC 0)
MS0,MI988=$00         ; All 16 ADCs unipolar
MS0,MI989=$8800       ; Card base address

// Setting up the automatic read function for the 2nd ACC-36E
MS0,MI1987=1          ; Enable automatic ADC read function (MACRO IC 1)
MS0,MI1988=$FF        ; All 16 ADCs bipolar
MS0,MI1989=$9800      ; Card base address
```



*Note*

A **Save** and a **\$\$\$** at the MACRO station are necessary to activate the automatic ADC processing: **MSSAV0**, followed by a **MS\$\$\$0**.

The automatic ADC processing copies the data into the following registers (\$20X) on the MACRO station:

1 <sup>st</sup> ACC-36E			
Channel	Location	Channel	Location
ADC1	Y:\$0200,0,12	ADC9	Y:\$0200,12,12
ADC2	Y:\$0201,0,12	ADC10	Y:\$0201,12,12
ADC3	Y:\$0202,0,12	ADC11	Y:\$0202,12,12
ADC4	Y:\$0203,0,12	ADC12	Y:\$0203,12,12
ADC5	Y:\$0204,0,12	ADC13	Y:\$0204,12,12
ADC6	Y:\$0205,0,12	ADC14	Y:\$0205,12,12
ADC7	Y:\$0206,0,12	ADC15	Y:\$0206,12,12
ADC8	Y:\$0207,0,12	ADC16	Y:\$0207,12,12

2 <sup>nd</sup> ACC-36E			
Channel	Location	Channel	Location
ADC1	Y:\$0208,0,12	ADC9	Y:\$0208,12,12
ADC2	Y:\$0209,0,12	ADC10	Y:\$0209,12,12
ADC3	Y:\$020A,0,12	ADC11	Y:\$020A,12,12
ADC4	Y:\$020B,0,12	ADC12	Y:\$020B,12,12
ADC5	Y:\$020C,0,12	ADC13	Y:\$020C,12,12
ADC6	Y:\$020D,0,12	ADC14	Y:\$020D,12,12
ADC7	Y:\$020E,0,12	ADC15	Y:\$020E,12,12
ADC8	Y:\$020F,0,12	ADC16	Y:\$020F,12,12

If the user desires, he or she can monitor the demuxed ADC values directly from the MACRO 16 CPU over MACRO ASCII communication using the following suggested MM-Variables (download to MACRO 16 CPU via MACRO ASCII communication) for troubleshooting purposes, or for use in a MACRO PLCC:

Suggested User MM-Variables			
1 <sup>st</sup> ACC-36E		2 <sup>nd</sup> ACC-36E	
#define	First36E_ADC1 MM1	#define	Second36E_ADC1 MM17
#define	First36E_ADC2 MM2	#define	Second36E_ADC2 MM18
#define	First36E_ADC3 MM3	#define	Second36E_ADC3 MM19
#define	First36E_ADC4 MM4	#define	Second36E_ADC4 MM20
#define	First36E_ADC5 MM5	#define	Second36E_ADC5 MM21
#define	First36E_ADC6 MM6	#define	Second36E_ADC6 MM22
#define	First36E_ADC7 MM7	#define	Second36E_ADC7 MM23
#define	First36E_ADC8 MM8	#define	Second36E_ADC8 MM24
#define	First36E_ADC9 MM9	#define	Second36E_ADC9 MM25
#define	First36E_ADC10 MM10	#define	Second36E_ADC10 MM26
#define	First36E_ADC11 MM11	#define	Second36E_ADC11 MM27
#define	First36E_ADC12 MM12	#define	Second36E_ADC12 MM28
#define	First36E_ADC13 MM13	#define	Second36E_ADC13 MM29
#define	First36E_ADC14 MM14	#define	Second36E_ADC14 MM30
#define	First36E_ADC15 MM15	#define	Second36E_ADC15 MM31
#define	First36E_ADC16 MM16	#define	Second36E_ADC16 MM32

	Unipolar	Bipolar
MACRO 16 CPU		
1 <sup>st</sup> ACC-36E	First36E_ADC1->Y:\$0200,0,12,U	First36E_ADC1->Y:\$0200,0,12,S
	First36E_ADC2->Y:\$0201,0,12,U	First36E_ADC2->Y:\$0201,0,12,S
	First36E_ADC3->Y:\$0202,0,12,U	First36E_ADC3->Y:\$0202,0,12,S
	First36E_ADC4->Y:\$0203,0,12,U	First36E_ADC4->Y:\$0203,0,12,S
	First36E_ADC5->Y:\$0204,0,12,U	First36E_ADC5->Y:\$0204,0,12,S
	First36E_ADC6->Y:\$0205,0,12,U	First36E_ADC6->Y:\$0205,0,12,S
	First36E_ADC7->Y:\$0206,0,12,U	First36E_ADC7->Y:\$0206,0,12,S
	First36E_ADC8->Y:\$0207,0,12,U	First36E_ADC8->Y:\$0207,0,12,S
	First36E_ADC9->Y:\$0200,12,12,U	First36E_ADC9->Y:\$0200,12,12,S
	First36E_ADC10->Y:\$0201,12,12,U	First36E_ADC10->Y:\$0201,12,12,S
	First36E_ADC11->Y:\$0202,12,12,U	First36E_ADC11->Y:\$0202,12,12,S
	First36E_ADC12->Y:\$0203,12,12,U	First36E_ADC12->Y:\$0203,12,12,S
	First36E_ADC13->Y:\$0204,12,12,U	First36E_ADC13->Y:\$0204,12,12,S
	First36E_ADC14->Y:\$0205,12,12,U	First36E_ADC14->Y:\$0205,12,12,S
	First36E_ADC15->Y:\$0206,12,12,U	First36E_ADC15->Y:\$0206,12,12,S
	First36E_ADC16->Y:\$0207,12,12,U	First36E_ADC16->Y:\$0207,12,12,S
2 <sup>nd</sup> ACC-36E	Second36E_ADC1->Y:\$0208,0,12,U	Second36E_ADC1->Y:\$0208,0,12,S
	Second36E_ADC2->Y:\$0209,0,12,U	Second36E_ADC2->Y:\$0209,0,12,S
	Second36E_ADC3->Y:\$020A,0,12,U	Second36E_ADC3->Y:\$020A,0,12,S
	Second36E_ADC4->Y:\$020B,0,12,U	Second36E_ADC4->Y:\$020B,0,12,S
	Second36E_ADC5->Y:\$020C,0,12,U	Second36E_ADC5->Y:\$020C,0,12,S
	Second36E_ADC6->Y:\$020D,0,12,U	Second36E_ADC6->Y:\$020D,0,12,S
	Second36E_ADC7->Y:\$020E,0,12,U	Second36E_ADC7->Y:\$020E,0,12,S
	Second36E_ADC8->Y:\$020F,0,12,U	Second36E_ADC8->Y:\$020F,0,12,S
	Second36E_ADC9->Y:\$0208,12,12,U	Second36E_ADC9->Y:\$0208,12,12,S
	Second36E_ADC10->Y:\$0209,12,12,U	Second36E_ADC10->Y:\$0209,12,12,S
	Second36E_ADC11->Y:\$020A,12,12,U	Second36E_ADC11->Y:\$020A,12,12,S
	Second36E_ADC12->Y:\$020B,12,12,U	Second36E_ADC12->Y:\$020B,12,12,S
	Second36E_ADC13->Y:\$020C,12,12,U	Second36E_ADC13->Y:\$020C,12,12,S
	Second36E_ADC14->Y:\$020D,12,12,U	Second36E_ADC14->Y:\$020D,12,12,S
	Second36E_ADC15->Y:\$020E,12,12,U	Second36E_ADC15->Y:\$020E,12,12,S
	Second36E_ADC16->Y:\$020F,12,12,U	Second36E_ADC16->Y:\$020F,12,12,S

## Transferring Data over I/O Nodes

The following explains the ADC data transfer from the \$020X to \$C0XX registers, and suggests user M-Variables for direct access.

It is assumed that the ADC processing has already been configured as explained in the previous step, and that the data is readily available in the \$020X registers. Also, it is assumed that communication over the MACRO ring has already been established, and that the user is familiar with node activation on both the Ring Controller and MACRO Station. Thus, any node(s) used in the following examples have to be enabled properly.

Having multiple cards of different types on the MACRO Station can make the I/O transfer management cumbersome. Therefore, two alternative methods are suggested:

- Automatic I/O node transfer, using MS{anynode}, MI173, MI174, and MI175  
This method is ideal for up to 2 ACC-36Es in one station.
- Manual I/O node transfer using MS{anynode}, MI20 through MI68.  
This method is recommended when more than two ACC-36E cards are present and/or there are other types of cards requiring a large amount of data transfer over I/O nodes. Note that this method can be used in conjunction with the automatic transfers.



### *Note*

This section assumes that MACRO ring, I/O nodes, and ring check error settings have been configured properly.

---

## Automatic I/O Node Data Transfer: MI173, MI174, MI175

The automatic I/O node transfer of ADCs is achieved using the following MACRO Station (IC#0) parameters:

MS{anynode},MI173 Copies lower 12 bits of up to 6 consecutive registers into six 16-bit nodes  
 MS{anynode},MI174 Copies upper 12 bits of up to 6 consecutive registers into six 16-bit nodes  
 MS{anynode},MI175 Copies lower & upper 12 bits (24 bits) of up to 2 consecutive 24-bit registers

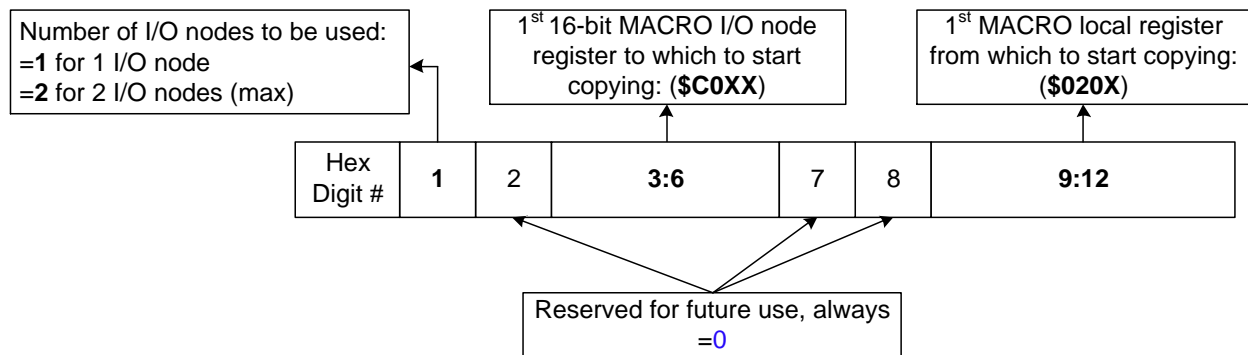


**Note**

For MACRO Station IC#1:

MS{anynode},MI173  
 MS{anynode},MI174  
 MS{anynode},MI175

These 48-bit variables are represented as 12 hexadecimal digits; they are set up as follows, where digit #1 is the leftmost digit when constructing the word:



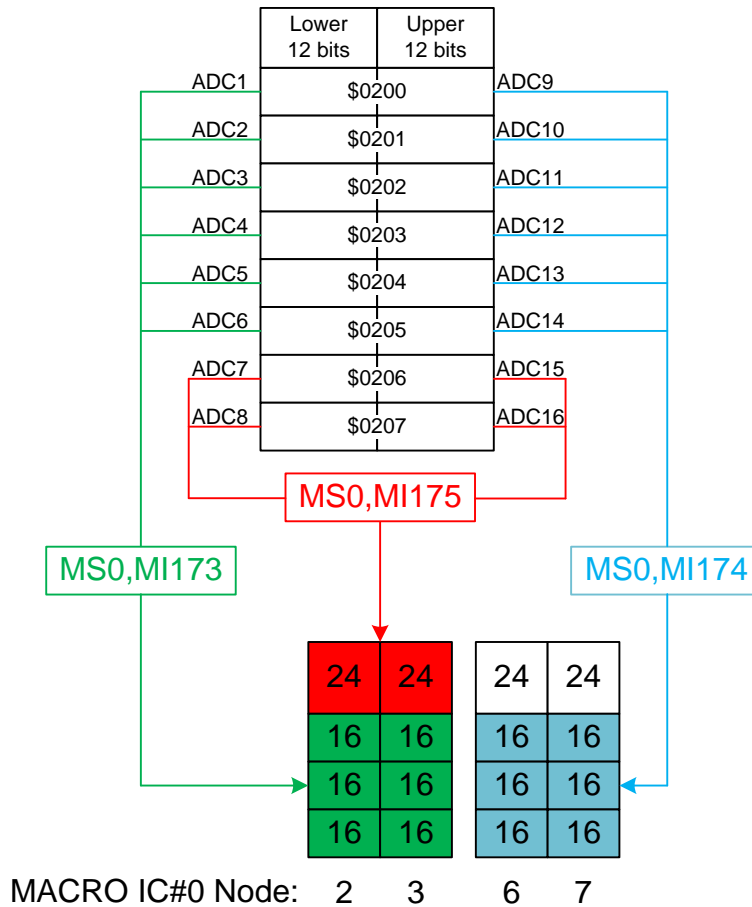
Following are two examples:

- Example 1: Automatic I/O node data transfer of 1 ACC-36E.
- Example 2: Automatic I/O node data transfer of 2 ACC-36Es.



**Example 1:** Setting up automatic I/O node transfer of 1 ACC-36E (total of 16 ADCs) using I/O nodes 2, 3, 6, and 7.

ADC1 through ADC6 will be copied to the six 16-bit registers of nodes 2 and 3.  
 ADC9 through ADC14 will be copied to the six 16-bit registers of nodes 6 and 7.  
 ADCs 7, 8, 15, 16 will be copied to the two 24-bit registers of nodes 2 and 3.



#### Example Code:

```
MS0,MI975=$CC ; MACRO IC#0 I/O node Enable, nodes 2, 3, 6, 7
MS0,MI19=4 ; MACRO Station I/O Data Transfer Period (adjustable)
MS0,MI173=$20C0A1000200 ; ADC1 thru ADC6 (lower 12 bits of $0200 thru $0205)
MS0,MI174=$20C0A9000200 ; ADC9 thru ADC14 (upper 12 bits of $0200 thru $0205)
MS0,MI175=$20C0A0000206 ; ADCs 7, 8, 15, 16 (lower + upper of $0206 thru $0207)
```

Suggested User M-Variables			
#define	First36E_ADC1	M5001	#define First36E_ADC9 M5009
#define	First36E_ADC2	M5002	#define First36E_ADC10 M5010
#define	First36E_ADC3	M5003	#define First36E_ADC11 M5011
#define	First36E_ADC4	M5004	#define First36E_ADC12 M5012
#define	First36E_ADC5	M5005	#define First36E_ADC13 M5013
#define	First36E_ADC6	M5006	#define First36E_ADC14 M5014
#define	First36E_ADC7	M5007	#define First36E_ADC15 M5015
#define	First36E_ADC8	M5008	#define First36E_ADC16 M5016

**Note**

These suggested variable numbers are for a Turbo Ultralite only. If the user has a Non-Turbo Ultralite, variables in the range of M0-M1023 must be used.

As configured by the ADC processing, the ADC pairs can be either unipolar (unsigned) or bipolar (signed):

Unipolar	Bipolar
Turbo PMAC2 Ultralite (or UMAC with ACC-5E)	
First36E_ADC1->X:\$078421,8,12	First36E_ADC1->X:\$078421,8,12,S
First36E_ADC2->X:\$078422,8,12	First36E_ADC2->X:\$078422,8,12,S
First36E_ADC3->X:\$078423,8,12	First36E_ADC3->X:\$078423,8,12,S
First36E_ADC4->X:\$078425,8,12	First36E_ADC4->X:\$078425,8,12,S
First36E_ADC5->X:\$078426,8,12	First36E_ADC5->X:\$078426,8,12,S
First36E_ADC6->X:\$078427,8,12	First36E_ADC6->X:\$078427,8,12,S
First36E_ADC7->X:\$078429,8,12	First36E_ADC7->X:\$078429,8,12,S
First36E_ADC8->X:\$07842A,8,12	First36E_ADC8->X:\$07842A,8,12,S
First36E_ADC9->X:\$07842B,8,12	First36E_ADC9->X:\$07842B,8,12,S
First36E_ADC10->X:\$07842D,8,12	First36E_ADC10->X:\$07842D,8,12,S
First36E_ADC11->X:\$07842E,8,12	First36E_ADC11->X:\$07842E,8,12,S
First36E_ADC12->X:\$07842F,8,12	First36E_ADC12->X:\$07842F,8,12,S
First36E_ADC13->X:\$078420,0,12	First36E_ADC13->X:\$078420,0,12,S
First36E_ADC14->X:\$078424,0,12	First36E_ADC14->X:\$078424,0,12,S
First36E_ADC15->X:\$078420,12,12	First36E_ADC15->X:\$078420,12,12,S
First36E_ADC16->X:\$078424,12,12	First36E_ADC16->X:\$078424,12,12,S
Non-Turbo PMAC2 Ultralite / MACRO Station Node Addresses	
First36E_ADC1->X:\$C0A1,8,12	First36E_ADC1->X:\$C0A1,8,12,S
First36E_ADC2->X:\$C0A2,8,12	First36E_ADC2->X:\$C0A2,8,12,S
First36E_ADC3->X:\$C0A3,8,12	First36E_ADC3->X:\$C0A3,8,12,S
First36E_ADC4->X:\$C0A5,8,12	First36E_ADC4->X:\$C0A5,8,12,S
First36E_ADC5->X:\$C0A6,8,12	First36E_ADC5->X:\$C0A6,8,12,S
First36E_ADC6->X:\$C0A7,8,12	First36E_ADC6->X:\$C0A7,8,12,S
First36E_ADC7->X:\$C0A9,8,12	First36E_ADC7->X:\$C0A9,8,12,S
First36E_ADC8->X:\$C0AA,8,12	First36E_ADC8->X:\$C0AA,8,12,S
First36E_ADC9->X:\$C0AB,8,12	First36E_ADC9->X:\$C0AB,8,12,S
First36E_ADC10->X:\$C0AD,8,12	First36E_ADC10->X:\$C0AD,8,12,S
First36E_ADC11->X:\$C0AE,8,12	First36E_ADC11->X:\$C0AE,8,12,S
First36E_ADC12->X:\$C0AF,8,12	First36E_ADC12->X:\$C0AF,8,12,S
First36E_ADC13->X:\$C0A0,0,12	First36E_ADC13->X:\$C0A0,0,12,S
First36E_ADC14->X:\$C0A4,0,12	First36E_ADC14->X:\$C0A4,0,12,S
First36E_ADC15->X:\$C0A0,12,12	First36E_ADC15->X:\$C0A0,12,12,S
First36E_ADC16->X:\$C0A4,12,12	First36E_ADC16->X:\$C0A4,12,12,S

**Example 2:** Setting up automatic I/O node transfer for two ACC-36Es (a total of 32 ADC channels).

First ACC-36E, using I/O nodes 2, 3, 6, and 7 of MACRO IC#0:

ADC1 through ADC6 will be copied to the six 16-bit registers of nodes 2 and 3 of MACRO IC#0.

ADC9 through ADC14 will be copied to the six 16-bit registers of nodes 6 and 7 of MACRO IC#0.

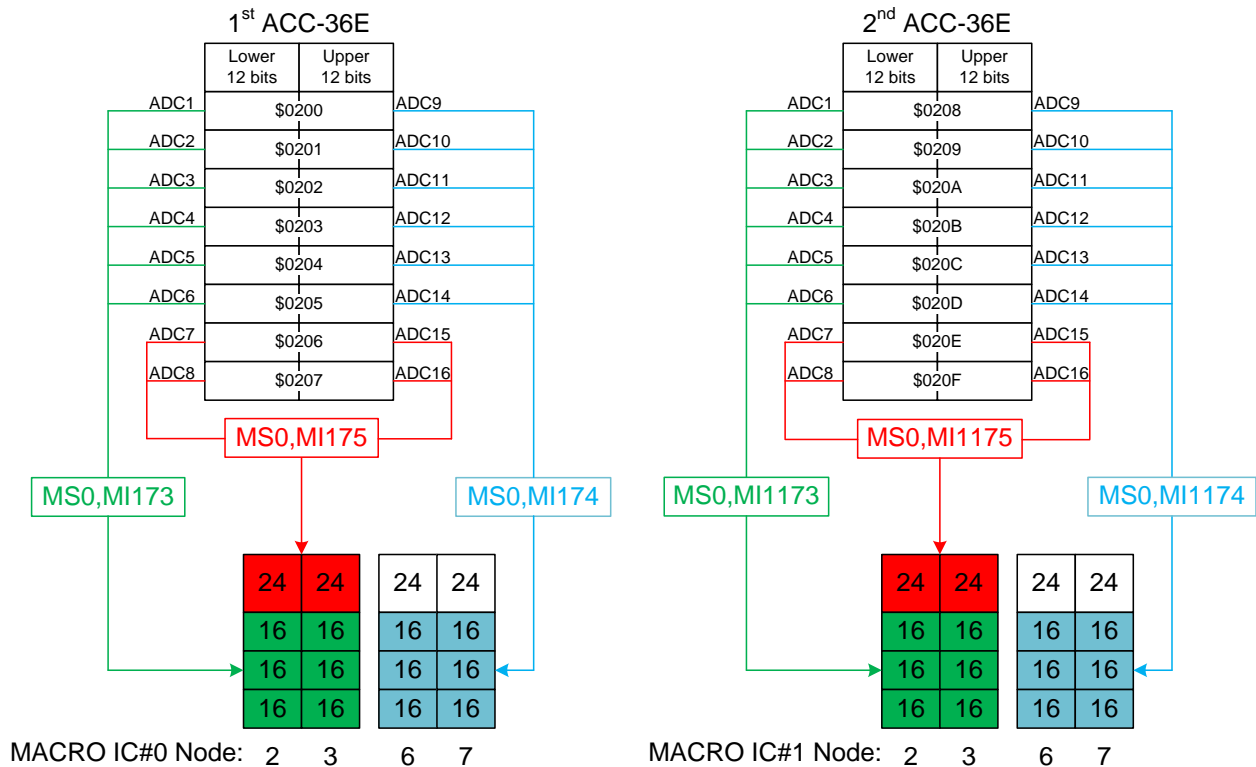
ADCs 7, 8, 15, 16 will be copied to the two 24-bit registers of nodes 2 and 3 of MACRO IC#0.

Second ACC-36E, using I/O nodes 2, 3, 6, and 7 of MACRO IC#1:

ADC1 through ADC6 will be copied to the six 16-bit registers of nodes 2 and 3 of MACRO IC#1.

ADC9 through ADC14 will be copied to the six 16-bit registers of nodes 6 and 7 of MACRO IC#1.

ADCs 7, 8, 15, 16 will be copied to the two 24-bit registers of nodes 2 and 3 of MACRO IC#1.



**Example Code:**

```

MS0,MI19=4 ; MACRO Station I/O Data Transfer Period (adjustable)

MS0,MI975=$CC ; MACRO IC#0 I/O Node Enable, nodes 2, 3, 6, 7

MS0,MI173=$20C0A1000200 ; 1st ACC-36E ADC1 thru ADC6 (lower 12 bits of $0200 thru $0205)
MS0,MI174=$20C0A9000200 ; 1st ACC-36E ADC9 thru ADC14 (upper 12 bits of $0200 thru $0205)
MS0,MI175=$20C0A0000206 ; 1st ACC-36E ADCs 7, 8, 15, 16 (lower + upper of $0206 thru $0207)

MS0,MI1975=$CC ; MACRO IC#1 I/O Node Enable, nodes 2, 3, 6, 7

MS0,MI1173=$20C0E1000208 ; 2nd ACC-36E ADC1 thru ADC6 (lower 12 bits of $0208 thru $020D)
MS0,MI1174=$20C0E9000208 ; 2nd ACC-36E ADC9 thru ADC14 (upper 12 bits of $0208 thru $020D)
MS0,MI1175=$20C0E000020E ; 2nd ACC-36E ADCs 7, 8, 15, 16 (lower + upper of $020E thru $020F)

```

Suggested User M-Variables			
1 <sup>st</sup> ACC-36E		2 <sup>nd</sup> ACC-36E	
#define	First36E_ADC1 M5001	#define	Second36E_ADC1 M5017
#define	First36E_ADC2 M5002	#define	Second36E_ADC2 M5018
#define	First36E_ADC3 M5003	#define	Second36E_ADC3 M5019
#define	First36E_ADC4 M5004	#define	Second36E_ADC4 M5020
#define	First36E_ADC5 M5005	#define	Second36E_ADC5 M5021
#define	First36E_ADC6 M5006	#define	Second36E_ADC6 M5022
#define	First36E_ADC7 M5007	#define	Second36E_ADC7 M5023
#define	First36E_ADC8 M5008	#define	Second36E_ADC8 M5024
#define	First36E_ADC9 M5009	#define	Second36E_ADC9 M5025
#define	First36E_ADC10 M5010	#define	Second36E_ADC10 M5026
#define	First36E_ADC11 M5011	#define	Second36E_ADC11 M5027
#define	First36E_ADC12 M5012	#define	Second36E_ADC12 M5028
#define	First36E_ADC13 M5013	#define	Second36E_ADC13 M5029
#define	First36E_ADC14 M5014	#define	Second36E_ADC14 M5030
#define	First36E_ADC15 M5015	#define	Second36E_ADC15 M5031
#define	First36E_ADC16 M5016	#define	Second36E_ADC16 M5032

**Note**

These suggested variable numbers are for a Turbo Ultralite only. If the user has a Non-Turbo Ultralite, variables in the range of M0-M1023 must be used.

As set up by the ADC processing, the ADC pairs can be either unipolar (unsigned) or bipolar (signed):

	Unipolar	Bipolar
<b>Turbo PMAC2 Ultralite (or UMAC with ACC-5E)</b>		
<b>1<sup>st</sup> ACC-36E</b>	First36E_ADC1->X:\$078421,8,12 First36E_ADC2->X:\$078422,8,12 First36E_ADC3->X:\$078423,8,12 First36E_ADC4->X:\$078425,8,12 First36E_ADC5->X:\$078426,8,12 First36E_ADC6->X:\$078427,8,12 First36E_ADC7->X:\$078429,8,12 First36E_ADC8->X:\$07842A,8,12 First36E_ADC9->X:\$07842B,8,12 First36E_ADC10->X:\$07842D,8,12 First36E_ADC11->X:\$07842E,8,12 First36E_ADC12->X:\$07842F,8,12 First36E_ADC13->X:\$078420,0,12 First36E_ADC14->X:\$078424,0,12 First36E_ADC15->X:\$078420,12,12 First36E_ADC16->X:\$078424,12,12	First36E_ADC1->X:\$078421,8,12,S First36E_ADC2->X:\$078422,8,12,S First36E_ADC3->X:\$078423,8,12,S First36E_ADC4->X:\$078425,8,12,S First36E_ADC5->X:\$078426,8,12,S First36E_ADC6->X:\$078427,8,12,S First36E_ADC7->X:\$078429,8,12,S First36E_ADC8->X:\$07842A,8,12,S First36E_ADC9->X:\$07842B,8,12,S First36E_ADC10->X:\$07842D,8,12,S First36E_ADC11->X:\$07842E,8,12,S First36E_ADC12->X:\$07842F,8,12,S First36E_ADC13->X:\$078420,0,12,S First36E_ADC14->X:\$078424,0,12,S First36E_ADC15->X:\$078420,12,12,S First36E_ADC16->X:\$078424,12,12,S
<b>2<sup>nd</sup> ACC-36E</b>	Second36E_ADC1->X:\$079421,8,12 Second36E_ADC2->X:\$079422,8,12 Second36E_ADC3->X:\$079423,8,12 Second36E_ADC4->X:\$079425,8,12 Second36E_ADC5->X:\$079426,8,12 Second36E_ADC6->X:\$079427,8,12 Second36E_ADC7->X:\$079429,8,12 Second36E_ADC8->X:\$07942A,8,12 Second36E_ADC9->X:\$07942B,8,12 Second36E_ADC10->X:\$07942D,8,12 Second36E_ADC11->X:\$07942E,8,12 Second36E_ADC12->X:\$07942F,8,12 Second36E_ADC13->X:\$079420,0,12 Second36E_ADC14->X:\$079424,0,12 Second36E_ADC15->X:\$079420,12,12 Second36E_ADC16->X:\$079424,12,12	Second36E_ADC1->X:\$079421,8,12,S Second36E_ADC2->X:\$079422,8,12,S Second36E_ADC3->X:\$079423,8,12,S Second36E_ADC4->X:\$079425,8,12,S Second36E_ADC5->X:\$079426,8,12,S Second36E_ADC6->X:\$079427,8,12,S Second36E_ADC7->X:\$079429,8,12,S Second36E_ADC8->X:\$07942A,8,12,S Second36E_ADC9->X:\$07942B,8,12,S Second36E_ADC10->X:\$07942D,8,12,S Second36E_ADC11->X:\$07942E,8,12,S Second36E_ADC12->X:\$07942F,8,12,S Second36E_ADC13->X:\$079420,0,12,S Second36E_ADC14->X:\$079424,0,12,S Second36E_ADC15->X:\$079420,12,12,S Second36E_ADC16->X:\$079424,12,12,S
<b>Non-Turbo PMAC2 Ultralite / MACRO Station Node Addresses</b>		
<b>1<sup>st</sup> ACC-36E</b>	First36E_ADC1->X:\$C0A1,8,12 First36E_ADC2->X:\$C0A2,8,12 First36E_ADC3->X:\$C0A3,8,12 First36E_ADC4->X:\$C0A5,8,12 First36E_ADC5->X:\$C0A6,8,12 First36E_ADC6->X:\$C0A7,8,12 First36E_ADC7->X:\$C0A9,8,12 First36E_ADC8->X:\$C0AA,8,12 First36E_ADC9->X:\$C0AB,8,12 First36E_ADC10->X:\$C0AD,8,12 First36E_ADC11->X:\$C0AE,8,12 First36E_ADC12->X:\$C0AF,8,12 First36E_ADC13->X:\$C0A0,0,12 First36E_ADC14->X:\$C0A4,0,12 First36E_ADC15->X:\$C0A0,12,12 First36E_ADC16->X:\$C0A4,12,12	First36E_ADC1->X:\$C0A1,8,12,S First36E_ADC2->X:\$C0A2,8,12,S First36E_ADC3->X:\$C0A3,8,12,S First36E_ADC4->X:\$C0A5,8,12,S First36E_ADC5->X:\$C0A6,8,12,S First36E_ADC6->X:\$C0A7,8,12,S First36E_ADC7->X:\$C0A9,8,12,S First36E_ADC8->X:\$C0AA,8,12,S First36E_ADC9->X:\$C0AB,8,12,S First36E_ADC10->X:\$C0AD,8,12,S First36E_ADC11->X:\$C0AE,8,12,S First36E_ADC12->X:\$C0AF,8,12,S First36E_ADC13->X:\$C0A0,0,12,S First36E_ADC14->X:\$C0A4,0,12,S First36E_ADC15->X:\$C0A0,12,12,S First36E_ADC16->X:\$C0A4,12,12,S
<b>2<sup>nd</sup> ACC-36E</b>	Second36E_ADC1->X:\$C0E1,8,12 Second36E_ADC2->X:\$C0E2,8,12 Second36E_ADC3->X:\$C0E3,8,12 Second36E_ADC4->X:\$C0E5,8,12 Second36E_ADC5->X:\$C0E6,8,12 Second36E_ADC6->X:\$C0E7,8,12 Second36E_ADC7->X:\$C0E9,8,12 Second36E_ADC8->X:\$C0EA,8,12 Second36E_ADC9->X:\$C0EB,8,12 Second36E_ADC10->X:\$C0ED,8,12 Second36E_ADC11->X:\$C0EE,8,12 Second36E_ADC12->X:\$C0EF,8,12 Second36E_ADC13->X:\$C0E0,0,12 Second36E_ADC14->X:\$C0E4,0,12 Second36E_ADC15->X:\$C0E0,12,12 Second36E_ADC16->X:\$C0E4,12,12	Second36E_ADC1->X:\$C0E1,8,12,S Second36E_ADC2->X:\$C0E2,8,12,S Second36E_ADC3->X:\$C0E3,8,12,S Second36E_ADC4->X:\$C0E5,8,12,S Second36E_ADC5->X:\$C0E6,8,12,S Second36E_ADC6->X:\$C0E7,8,12,S Second36E_ADC7->X:\$C0E9,8,12,S Second36E_ADC8->X:\$C0EA,8,12,S Second36E_ADC9->X:\$C0EB,8,12,S Second36E_ADC10->X:\$C0ED,8,12,S Second36E_ADC11->X:\$C0EE,8,12,S Second36E_ADC12->X:\$C0EF,8,12,S Second36E_ADC13->X:\$C0E0,0,12,S Second36E_ADC14->X:\$C0E4,0,12,S Second36E_ADC15->X:\$C0E0,12,12,S Second36E_ADC16->X:\$C0E4,12,12,S

## Manual I/O Node Data Transfer: MI19...MI68

The manual I/O node transfer of ADCs is achieved using the following MACRO Station parameters:

- **MS{anynode},MI19:** I/O data transfer period.

MI19 controls the data transfer period (in phase cycles) between the MACRO node interface registers and the I/O registers. If MI19 is set to 0, the data transfer is disabled.

MI19 is typically set to 4 phase cycles.

- **MS{anynode},MI20:** Data transfer enable mask.

MI20 controls which of 48 possible data transfer operations (specified by MI21-MI68) are performed at the data transfer period set by MI19.

MI20 is a 48-bit value; each bit controls whether the data transfer specified by one of the variables MI21 through MI68 is performed.

### Examples:

If MI20=\$1 → bit #0 is set to 1 → MI21 transfer is performed.

If MI20=\$3 → bits #0 and #1 are set to 1 → MI21 and MI22 are performed.

If MI20=\$5 → bits #0 and #2 are set to 1 → MI21 and MI23 are performed.

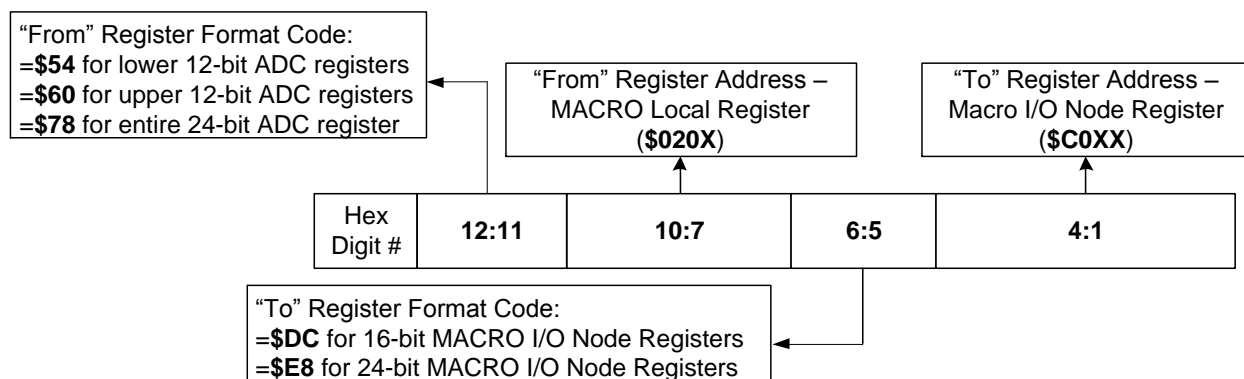
If MI20=\$F → bits #0 through #3 are set to 1 → MI21 through MI24 are performed.

- **MS{anynode},MI21-MI68:** Data transfer source and destination addresses.

MI21-MI68 each specify a type of data transfer (copying) operation that will occur on the MACRO Station at a rate specified by Station Variable MI19, and enabled by Station variable MI20.

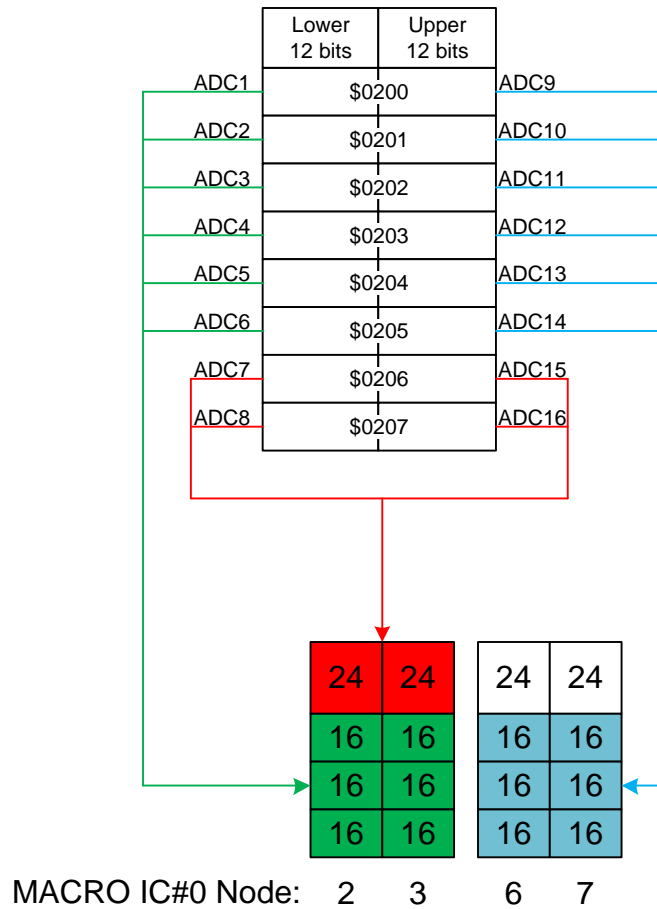
Each variable also specifies the address from which the data will be copied (read), and the address to which the data will be copied (written).

MI21-MI68 are 48-bit variables, represented as 12 hexadecimal digits as follows, where digit #1 is the rightmost digit when constructing the word:



Following are two examples:

- Example 1: Manual I/O node data transfer of 1 ACC-36E.
- Example 2: Manual I/O node data transfer of 2 ACC-36Es.

**Example 1: One ACC-36E at \$8800, Bipolar.**

```

//***** ACC-36E Manual MACRO I/O Transfer Example *****/
// Uses MI19, MI20, and MI21-MI34
// Configures 1 ACC-36E at base address $78C00, all Bipolar

// Setting up the automatic read function for the 1st ACC-36E
MS0,MI987 = 1      ; Enable Automatic ADC Demuxing
MS0,MI988 = $FF    ; All 16 ADCs Bipolar
MS0,MI989 = $8800  ; Card base address
MS0,MI975 = $CC    ; MACRO IC#0 I/O node Enable, nodes 2, 3, 6, 7

MS0,MI19 = 4       ; Perform the transfer every 4 phase cycles
MS0,MI20 = $3FFF   ; Use 14 I/O transfer variables every MI19 phase cycles (use MI21-MI34)

// For the 1st ACC-36E (on MACRO IC#0)
MS0,MI21=$540200DCC0A1 ; ADC 1 goes to 1st 16-bit register of Node 2
MS0,MI22=$600200DCC0A9 ; ADC 9 goes to 1st 16-bit register of Node 6
MS0,MI23=$540201DCC0A2 ; ADC 2 goes to 2nd 16-bit register of Node 2
MS0,MI24=$600201DCC0AA ; ADC 10 goes to 2nd 16-bit register of Node 6
MS0,MI25=$540202DCC0A3 ; ADC 3 goes to 3rd 16-bit register of Node 2
MS0,MI26=$600202DCC0AB ; ADC 11 goes to 3rd 16-bit register of Node 6
MS0,MI27=$540203DCC0A5 ; ADC 4 goes to 1st 16-bit register of Node 3
MS0,MI28=$600203DCC0AD ; ADC 12 goes to 1st 16-bit register of Node 7
MS0,MI29=$540204DCC0A6 ; ADC 5 goes to 2nd 16-bit register of Node 3
MS0,MI30=$600204DCC0AE ; ADC 13 goes to 2nd 16-bit register of Node 7
MS0,MI31=$540205DCC0A7 ; ADC 6 goes to 3rd 16-bit register of Node 3
MS0,MI32=$600205DCC0AF ; ADC 14 goes to 3rd 16-bit register of Node 7
MS0,MI33=$780206E8C0A0 ; ADCs 7 and 15 go to 24-bit register of Node 2
MS0,MI34=$780207E8C0A4 ; ADCs 8 and 16 go to 24-bit register of Node 3

```

Suggested User M-Variables:

1 <sup>st</sup> ACC-36E		
#define	First36E_ADC1	M5001
#define	First36E_ADC2	M5002
#define	First36E_ADC3	M5003
#define	First36E_ADC4	M5004
#define	First36E_ADC5	M5005
#define	First36E_ADC6	M5006
#define	First36E_ADC7	M5007
#define	First36E_ADC8	M5008
#define	First36E_ADC9	M5009
#define	First36E_ADC10	M5010
#define	First36E_ADC11	M5011
#define	First36E_ADC12	M5012
#define	First36E_ADC13	M5013
#define	First36E_ADC14	M5014
#define	First36E_ADC15	M5015
#define	First36E_ADC16	M5016



*Note*

These suggested variable numbers are for a Turbo Ultralite only. If the user has a Non-Turbo Ultralite, variables in the range of M0-M1023 must be used.

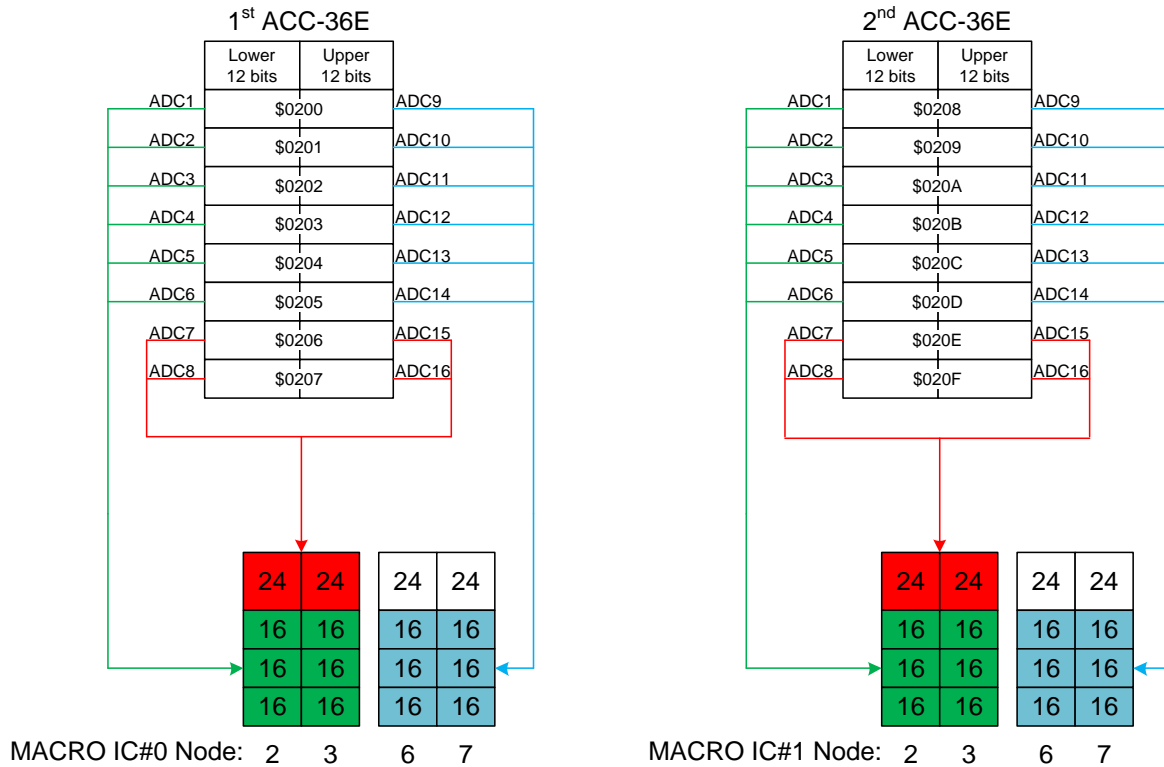
---



As set up by the ADC processing, the ADC pairs can be either unipolar (unsigned) or bipolar (signed):

	Unipolar	Bipolar
<b>Turbo PMAC2 Ultralite (or UMAC with ACC-5E)</b>		
<b>1<sup>st</sup> ACC-36E</b>	First36E_ADC1->X:\$078421,8,12 First36E_ADC2->X:\$078422,8,12 First36E_ADC3->X:\$078423,8,12 First36E_ADC4->X:\$078425,8,12 First36E_ADC5->X:\$078426,8,12 First36E_ADC6->X:\$078427,8,12 First36E_ADC7->X:\$078429,8,12 First36E_ADC8->X:\$07842A,8,12 First36E_ADC9->X:\$07842B,8,12 First36E_ADC10->X:\$07842D,8,12 First36E_ADC11->X:\$07842E,8,12 First36E_ADC12->X:\$07842F,8,12 First36E_ADC13->X:\$078420,0,12 First36E_ADC14->X:\$078424,0,12 First36E_ADC15->X:\$078420,12,12 First36E_ADC16->X:\$078424,12,12	First36E_ADC1->X:\$078421,8,12,S First36E_ADC2->X:\$078422,8,12,S First36E_ADC3->X:\$078423,8,12,S First36E_ADC4->X:\$078425,8,12,S First36E_ADC5->X:\$078426,8,12,S First36E_ADC6->X:\$078427,8,12,S First36E_ADC7->X:\$078429,8,12,S First36E_ADC8->X:\$07842A,8,12,S First36E_ADC9->X:\$07842B,8,12,S First36E_ADC10->X:\$07842D,8,12,S First36E_ADC11->X:\$07842E,8,12,S First36E_ADC12->X:\$07842F,8,12,S First36E_ADC13->X:\$078420,0,12,S First36E_ADC14->X:\$078424,0,12,S First36E_ADC15->X:\$078420,12,12,S First36E_ADC16->X:\$078424,12,12,S
<b>Non-Turbo PMAC2 Ultralite / MACRO Station Node Addresses</b>		
<b>1<sup>st</sup> ACC-36E</b>	First36E_ADC1->X:\$C0A1,8,12 First36E_ADC2->X:\$C0A2,8,12 First36E_ADC3->X:\$C0A3,8,12 First36E_ADC4->X:\$C0A5,8,12 First36E_ADC5->X:\$C0A6,8,12 First36E_ADC6->X:\$C0A7,8,12 First36E_ADC7->X:\$C0A9,8,12 First36E_ADC8->X:\$C0AA,8,12 First36E_ADC9->X:\$C0AB,8,12 First36E_ADC10->X:\$C0AD,8,12 First36E_ADC11->X:\$C0AE,8,12 First36E_ADC12->X:\$C0AF,8,12 First36E_ADC13->X:\$C0A0,0,12 First36E_ADC14->X:\$C0A4,0,12 First36E_ADC15->X:\$C0A0,12,12 First36E_ADC16->X:\$C0A4,12,12	First36E_ADC1->X:\$C0A1,8,12,S First36E_ADC2->X:\$C0A2,8,12,S First36E_ADC3->X:\$C0A3,8,12,S First36E_ADC4->X:\$C0A5,8,12,S First36E_ADC5->X:\$C0A6,8,12,S First36E_ADC6->X:\$C0A7,8,12,S First36E_ADC7->X:\$C0A9,8,12,S First36E_ADC8->X:\$C0AA,8,12,S First36E_ADC9->X:\$C0AB,8,12,S First36E_ADC10->X:\$C0AD,8,12,S First36E_ADC11->X:\$C0AE,8,12,S First36E_ADC12->X:\$C0AF,8,12,S First36E_ADC13->X:\$C0A0,0,12,S First36E_ADC14->X:\$C0A4,0,12,S First36E_ADC15->X:\$C0A0,12,12,S First36E_ADC16->X:\$C0A4,12,12,S

**Example 2:** 1st ACC-36E at \$8800, Unipolar, and 2<sup>nd</sup> ACC-36E at \$9800, Bipolar.



```

//***** ACC-36E Manual MACRO I/O Transfer Example *****/
// Uses MI19, MI20, and MI21-MI48
// Configures 1st ACC-36E at base address $78C00, all Unipolar,
// and configures 2nd ACC-36E at base address $79C00, all Bipolar

// Setting up the automatic read function for the 1st ACC-36E
MS0,MI987 = 1      ; Enable Automatic ADC Demuxing
MS0,MI988 = $00    ; All 16 ADCs Unipolar
MS0,MI989 = $8800  ; Card base address
MS0,MI975 = $CC    ; MACRO IC#0 I/O node Enable, nodes 2, 3, 6, 7

// Setting up the automatic read function for the 2nd ACC-36E
MS0,MI1987=1      ; Enable automatic ADC read function (MACRO IC 1)
MS0,MI1988=$FF    ; All 16 ADCs Bipolar
MS0,MI1989=$9800  ; Card base address
MS0,MI1975=$CC    ; MACRO IC#1 I/O node Enable, nodes 2, 3, 6, 7

MS0,MI19 = 4      ; Perform the transfer every 4 phase cycles
MS0,MI20 = $FFFFFF ; Use 28 I/O transfer variables every MI19 phase cycles (use MI21-MI48)

// For the 1st ACC-36E (on MACRO IC#0)
MS0,MI21=$540200DCC0A1 ; ADC 1 goes to 1st 16-bit register of Node 2
MS0,MI22=$600200DCC0A9 ; ADC 9 goes to 1st 16-bit register of Node 6
MS0,MI23=$540201DCC0A2 ; ADC 2 goes to 2nd 16-bit register of Node 2
MS0,MI24=$600201DCC0AA ; ADC 10 goes to 2nd 16-bit register of Node 6
MS0,MI25=$540202DCC0A3 ; ADC 3 goes to 3rd 16-bit register of Node 2
MS0,MI26=$600202DCC0AB ; ADC 11 goes to 3rd 16-bit register of Node 6
MS0,MI27=$540203DCC0A5 ; ADC 4 goes to 1st 16-bit register of Node 3
MS0,MI28=$600203DCC0AD ; ADC 12 goes to 1st 16-bit register of Node 7
MS0,MI29=$540204DCC0A6 ; ADC 5 goes to 2nd 16-bit register of Node 3
MS0,MI30=$600204DCC0AE ; ADC 13 goes to 2nd 16-bit register of Node 7
MS0,MI31=$540205DCC0A7 ; ADC 6 goes to 3rd 16-bit register of Node 3
MS0,MI32=$600205DCC0AF ; ADC 14 goes to 3rd 16-bit register of Node 7
MS0,MI33=$780206E8C0A0 ; ADCs 7 and 15 go to 24-bit register of Node 2
MS0,MI34=$780207E8C0A4 ; ADCs 8 and 16 go to 24-bit register of Node 3

```

```
// For the 2nd ACC-36E (on MACRO IC#1)
MS0,MI35=$540208DCC0E1 ; ADC 1 goes to 1st 16-bit register of Node 2
MS0,MI36=$600208DCC0E9 ; ADC 9 goes to 1st 16-bit register of Node 6
MS0,MI37=$540209DCC0E2 ; ADC 2 goes to 2nd 16-bit register of Node 2
MS0,MI38=$600209DCC0EA ; ADC 10 goes to 2nd 16-bit register of Node 6
MS0,MI39=$54020ADCC0E3 ; ADC 3 goes to 3rd 16-bit register of Node 2
MS0,MI40=$60020ADCC0EB ; ADC 11 goes to 3rd 16-bit register of Node 6
MS0,MI41=$54020BDCC0E5 ; ADC 4 goes to 1st 16-bit register of Node 3
MS0,MI42=$60020BDCC0ED ; ADC 12 goes to 1st 16-bit register of Node 7
MS0,MI43=$54020CDCC0E6 ; ADC 5 goes to 2nd 16-bit register of Node 3
MS0,MI44=$60020CDCC0EE ; ADC 13 goes to 2nd 16-bit register of Node 7
MS0,MI45=$54020DDCC0E7 ; ADC 6 goes to 3rd 16-bit register of Node 3
MS0,MI46=$60020DDCC0EF ; ADC 14 goes to 3rd 16-bit register of Node 7
MS0,MI47=$78020EE8C0E0 ; ADCs 7 and 15 go to 24-bit register of Node 2
MS0,MI48=$78020FE8C0E4 ; ADCs 8 and 16 go to 24-bit register of Node 3
```

Suggested User M-Variables			
1 <sup>st</sup> ACC-36E		2 <sup>nd</sup> ACC-36E	
#define	First36E_ADC1 M5001	#define	Second36E_ADC1 M5017
#define	First36E_ADC2 M5002	#define	Second36E_ADC2 M5018
#define	First36E_ADC3 M5003	#define	Second36E_ADC3 M5019
#define	First36E_ADC4 M5004	#define	Second36E_ADC4 M5020
#define	First36E_ADC5 M5005	#define	Second36E_ADC5 M5021
#define	First36E_ADC6 M5006	#define	Second36E_ADC6 M5022
#define	First36E_ADC7 M5007	#define	Second36E_ADC7 M5023
#define	First36E_ADC8 M5008	#define	Second36E_ADC8 M5024
#define	First36E_ADC9 M5009	#define	Second36E_ADC9 M5025
#define	First36E_ADC10 M5010	#define	Second36E_ADC10 M5026
#define	First36E_ADC11 M5011	#define	Second36E_ADC11 M5027
#define	First36E_ADC12 M5012	#define	Second36E_ADC12 M5028
#define	First36E_ADC13 M5013	#define	Second36E_ADC13 M5029
#define	First36E_ADC14 M5014	#define	Second36E_ADC14 M5030
#define	First36E_ADC15 M5015	#define	Second36E_ADC15 M5031
#define	First36E_ADC16 M5016	#define	Second36E_ADC16 M5032

**Note**

These suggested variable numbers are for a Turbo Ultralite only. If the user has a Non-Turbo Ultralite, variables in the range of M0-M1023 must be used.

As set up by the ADC processing, the ADC pairs can be either unipolar (unsigned) or bipolar (signed):

	Unipolar	Bipolar
<b>Turbo PMAC2 Ultralite (or UMAC with ACC-5E)</b>		
<b>1<sup>st</sup> ACC-36E</b>	First36E_ADC1->X:\$078421,8,12 First36E_ADC2->X:\$078422,8,12 First36E_ADC3->X:\$078423,8,12 First36E_ADC4->X:\$078425,8,12 First36E_ADC5->X:\$078426,8,12 First36E_ADC6->X:\$078427,8,12 First36E_ADC7->X:\$078429,8,12 First36E_ADC8->X:\$07842A,8,12 First36E_ADC9->X:\$07842B,8,12 First36E_ADC10->X:\$07842D,8,12 First36E_ADC11->X:\$07842E,8,12 First36E_ADC12->X:\$07842F,8,12 First36E_ADC13->X:\$078420,0,12 First36E_ADC14->X:\$078424,0,12 First36E_ADC15->X:\$078420,12,12 First36E_ADC16->X:\$078424,12,12	First36E_ADC1->X:\$078421,8,12,S First36E_ADC2->X:\$078422,8,12,S First36E_ADC3->X:\$078423,8,12,S First36E_ADC4->X:\$078425,8,12,S First36E_ADC5->X:\$078426,8,12,S First36E_ADC6->X:\$078427,8,12,S First36E_ADC7->X:\$078429,8,12,S First36E_ADC8->X:\$07842A,8,12,S First36E_ADC9->X:\$07842B,8,12,S First36E_ADC10->X:\$07842D,8,12,S First36E_ADC11->X:\$07842E,8,12,S First36E_ADC12->X:\$07842F,8,12,S First36E_ADC13->X:\$078420,0,12,S First36E_ADC14->X:\$078424,0,12,S First36E_ADC15->X:\$078420,12,12,S First36E_ADC16->X:\$078424,12,12,S
<b>2<sup>nd</sup> ACC-36E</b>	Second36E_ADC1->X:\$079421,8,12 Second36E_ADC2->X:\$079422,8,12 Second36E_ADC3->X:\$079423,8,12 Second36E_ADC4->X:\$079425,8,12 Second36E_ADC5->X:\$079426,8,12 Second36E_ADC6->X:\$079427,8,12 Second36E_ADC7->X:\$079429,8,12 Second36E_ADC8->X:\$07942A,8,12 Second36E_ADC9->X:\$07942B,8,12 Second36E_ADC10->X:\$07942D,8,12 Second36E_ADC11->X:\$07942E,8,12 Second36E_ADC12->X:\$07942F,8,12 Second36E_ADC13->X:\$079420,0,12 Second36E_ADC14->X:\$079424,0,12 Second36E_ADC15->X:\$079420,12,12 Second36E_ADC16->X:\$079424,12,12	Second36E_ADC1->X:\$079421,8,12,S Second36E_ADC2->X:\$079422,8,12,S Second36E_ADC3->X:\$079423,8,12,S Second36E_ADC4->X:\$079425,8,12,S Second36E_ADC5->X:\$079426,8,12,S Second36E_ADC6->X:\$079427,8,12,S Second36E_ADC7->X:\$079429,8,12,S Second36E_ADC8->X:\$07942A,8,12,S Second36E_ADC9->X:\$07942B,8,12,S Second36E_ADC10->X:\$07942D,8,12,S Second36E_ADC11->X:\$07942E,8,12,S Second36E_ADC12->X:\$07942F,8,12,S Second36E_ADC13->X:\$079420,0,12,S Second36E_ADC14->X:\$079424,0,12,S Second36E_ADC15->X:\$079420,12,12,S Second36E_ADC16->X:\$079424,12,12,S
<b>Non-Turbo PMAC2 Ultralite / MACRO Station Node Addresses</b>		
<b>1<sup>st</sup> ACC-36E</b>	First36E_ADC1->X:\$C0A1,8,12 First36E_ADC2->X:\$C0A2,8,12 First36E_ADC3->X:\$C0A3,8,12 First36E_ADC4->X:\$C0A5,8,12 First36E_ADC5->X:\$C0A6,8,12 First36E_ADC6->X:\$C0A7,8,12 First36E_ADC7->X:\$C0A9,8,12 First36E_ADC8->X:\$C0AA,8,12 First36E_ADC9->X:\$C0AB,8,12 First36E_ADC10->X:\$C0AD,8,12 First36E_ADC11->X:\$C0AE,8,12 First36E_ADC12->X:\$C0AF,8,12 First36E_ADC13->X:\$C0A0,0,12 First36E_ADC14->X:\$C0A4,0,12 First36E_ADC15->X:\$C0A0,12,12 First36E_ADC16->X:\$C0A4,12,12	First36E_ADC1->X:\$C0A1,8,12,S First36E_ADC2->X:\$C0A2,8,12,S First36E_ADC3->X:\$C0A3,8,12,S First36E_ADC4->X:\$C0A5,8,12,S First36E_ADC5->X:\$C0A6,8,12,S First36E_ADC6->X:\$C0A7,8,12,S First36E_ADC7->X:\$C0A9,8,12,S First36E_ADC8->X:\$C0AA,8,12,S First36E_ADC9->X:\$C0AB,8,12,S First36E_ADC10->X:\$C0AD,8,12,S First36E_ADC11->X:\$C0AE,8,12,S First36E_ADC12->X:\$C0AF,8,12,S First36E_ADC13->X:\$C0A0,0,12,S First36E_ADC14->X:\$C0A4,0,12,S First36E_ADC15->X:\$C0A0,12,12,S First36E_ADC16->X:\$C0A4,12,12,S
<b>2<sup>nd</sup> ACC-36E</b>	Second36E_ADC1->X:\$C0E1,8,12 Second36E_ADC2->X:\$C0E2,8,12 Second36E_ADC3->X:\$C0E3,8,12 Second36E_ADC4->X:\$C0E5,8,12 Second36E_ADC5->X:\$C0E6,8,12 Second36E_ADC6->X:\$C0E7,8,12 Second36E_ADC7->X:\$C0E9,8,12 Second36E_ADC8->X:\$C0EA,8,12 Second36E_ADC9->X:\$C0EB,8,12 Second36E_ADC10->X:\$C0ED,8,12 Second36E_ADC11->X:\$C0EE,8,12 Second36E_ADC12->X:\$C0EF,8,12 Second36E_ADC13->X:\$C0E0,0,12 Second36E_ADC14->X:\$C0E4,0,12 Second36E_ADC15->X:\$C0E0,12,12 Second36E_ADC16->X:\$C0E4,12,12	Second36E_ADC1->X:\$C0E1,8,12,S Second36E_ADC2->X:\$C0E2,8,12,S Second36E_ADC3->X:\$C0E3,8,12,S Second36E_ADC4->X:\$C0E5,8,12,S Second36E_ADC5->X:\$C0E6,8,12,S Second36E_ADC6->X:\$C0E7,8,12,S Second36E_ADC7->X:\$C0E9,8,12,S Second36E_ADC8->X:\$C0EA,8,12,S Second36E_ADC9->X:\$C0EB,8,12,S Second36E_ADC10->X:\$C0ED,8,12,S Second36E_ADC11->X:\$C0EE,8,12,S Second36E_ADC12->X:\$C0EF,8,12,S Second36E_ADC13->X:\$C0E0,0,12,S Second36E_ADC14->X:\$C0E4,0,12,S Second36E_ADC15->X:\$C0E0,12,12,S Second36E_ADC16->X:\$C0E4,12,12,S

## Using an Analog Input for Servo Feedback over MACRO

The ACC-36E analog inputs can be used as a feedback device for a servo motor, even over MACRO.



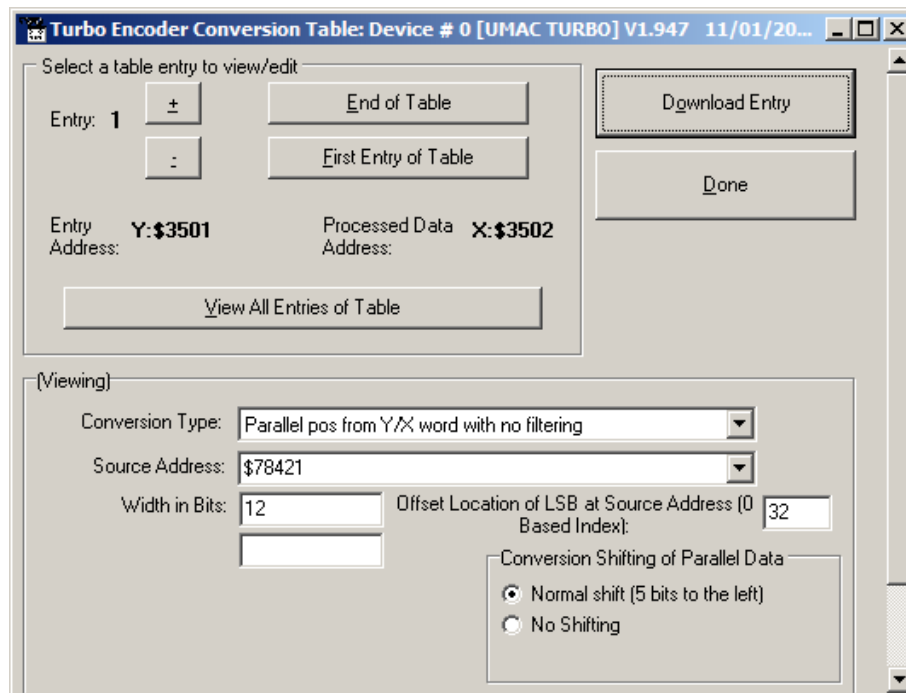
**Note**

- Refer to Delta Tau's released application notes or Turbo User Manual for cascaded-loop control (i.e. force, height control around position loop).
- The automatic ADC read function is recommended for this application.

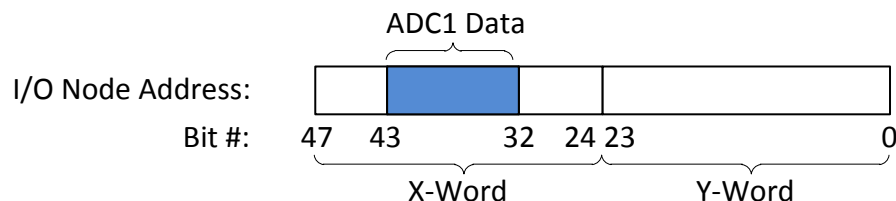
### Example:

Setting up Motor #1 position and velocity feedback from an ADC channel coming over the MACRO ring on MACRO IC#0, Node 2, 1<sup>st</sup> 16-bit register (bits 8-19 of X:\$78421).

The analog input is brought into the Encoder Conversion Table as a Parallel Y/X word with no filtering:



The Offset Location of LSB at Source Address is set to 32 because this type of ECT entry begins at bit 0 of the Y-word (24 bits wide) of the I/O Node Address. So, to get to bit 8 of the X-word, one must offset the LSB by 32 (=24+8):



The equivalent code in Turbo PMAC Encoder Conversion Table parameters:

I8000=\$78421	; Unfiltered parallel pos of location X:\$78421
I8001=\$00C020	; Width and Offset.

The position and velocity pointers are then set to the processed data address (i.e. \$3502):

I103=\$3502	; Motor #1 position loop feedback address
I104=\$3502	; Motor #1 velocity loop feedback address



If “No Shifting” is used in this example (see “Conversion Shifting of Parallel Data” in the screenshot on the preceding page), the result of the Encoder Conversion Table entry (in the X:\$3502 register in this example) must be multiplied by 32 ( $=2^5$ ) in order to obtain the same value stored in bits 8-19 of X:\$78421.

## Analog Input Power-On Position over MACRO

Some analog devices are absolute along the travel range of the motor (e.g., in hydraulic piston applications). Generally, it is desirable to obtain the motor position (input voltage) on power up or reset. This procedure can be done in a simple PLC on power-up by writing the processed A/D data into the motor actual position register (suggested M-Variable Mxx62).



- If the automatic ADC read method is being used, waiting a delay of about ½ second after the PMAC boots should be allowed for processing the data before copying it into the motor actual position register.
- If the manual ADC read method is being used, it is recommended to add this procedure at the end of the manual read PLC, or in a subsequent PLC with ½ sec delay to allow data processing.

**Example:** Reading Motor #1 position on power-up or reset, assuming that the automatic read function is used, and that M5001 is predefined to read bits 8-19 of the 1<sup>st</sup> 16-bit register of Node 2 of MACRO IC#0 where the result of the first ACC-36E’s ADC channel 1 (unipolar) resides.

```
End Gat
Del Gat
Close

#define MtrlActPos          M162      ; Motor #1 Actual Position
                                ; Suggested M-Variable units of 1/32*I108
MtrlActPos->D:$8B

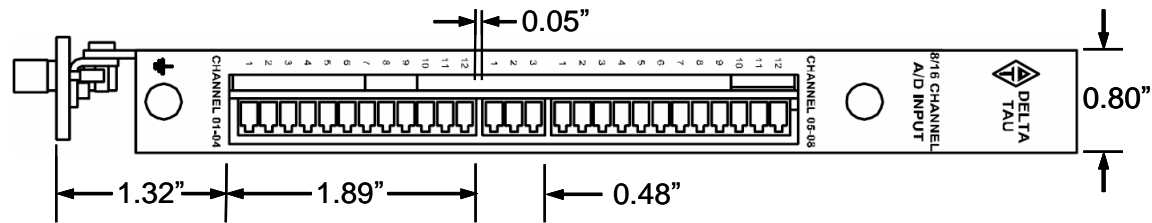
#define First36E_ADC1      M5001     ; Channel 1 ADC
First36E_ADC1->X:$078421,8,12,U    ; Unipolar

Open PLC 1 Clear
I5111=500*8388608/I10
While (I5111>0) EndWhile           ; ½ sec delay
MtrlActPos=First36E_ADC1*32*I108    ; Motor #1 Actual Position (scaled to motor counts)
Disable PLC 1                      ; Scan once on power-up or reset
Close
```

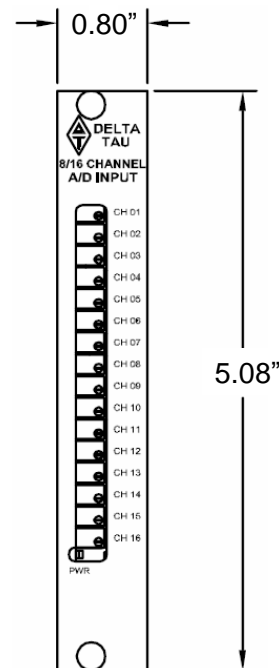
## ACC-36E LAYOUT & PINOUTS

### Terminal Block Option

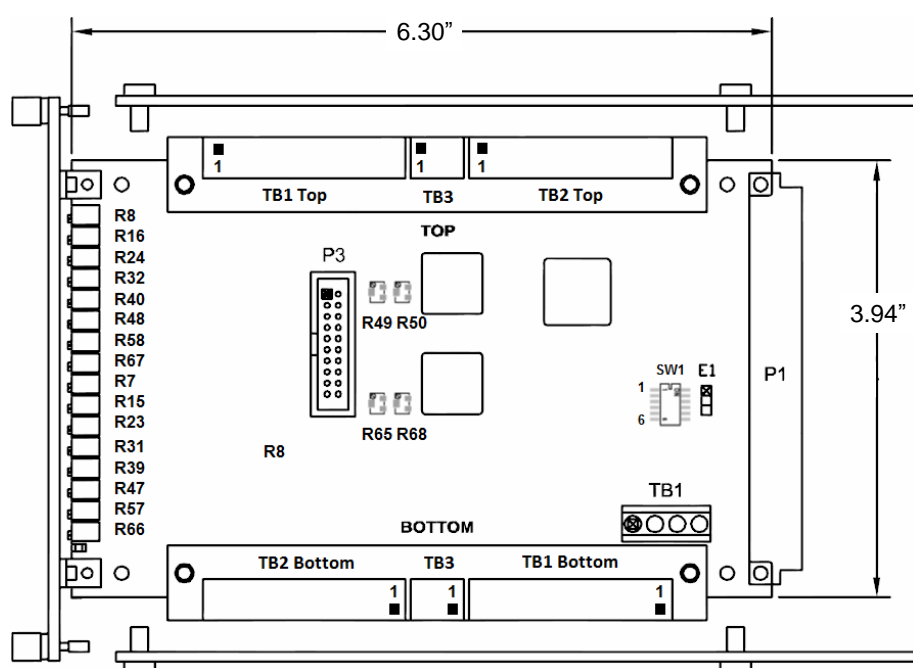
#### Top View



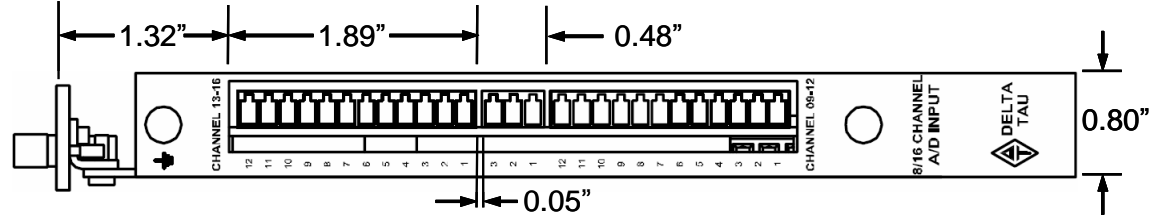
#### Front View



#### Side View



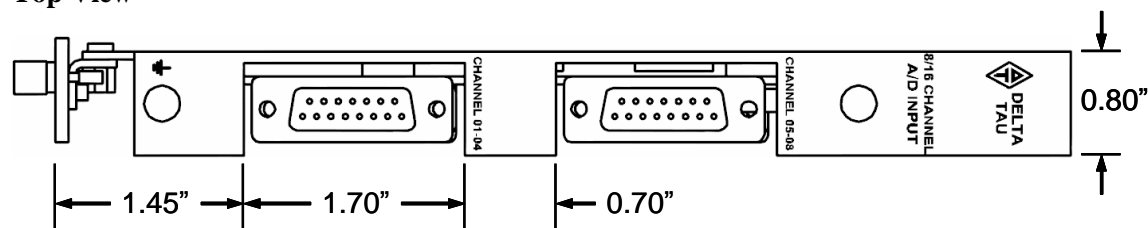
#### Bottom View



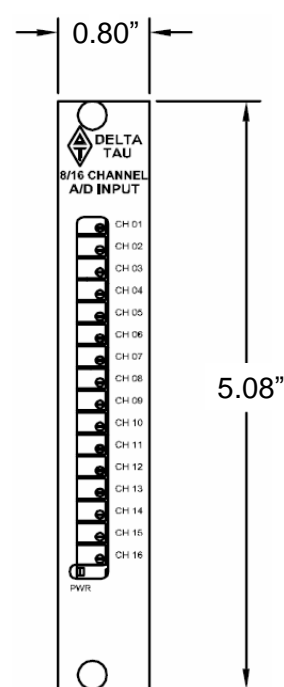
All dimensions are in inches. Drawings are not to scale.

## D-Sub Option

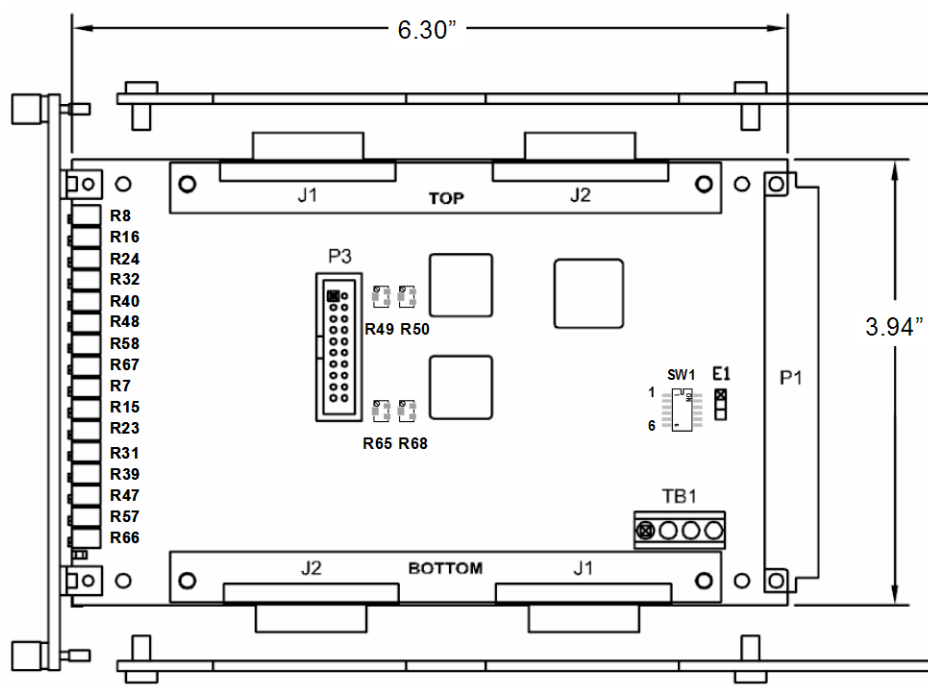
### Top View



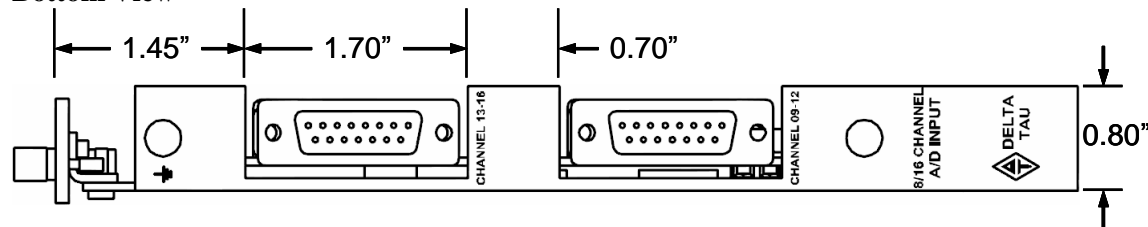
### Front View



### Side View



### Bottom View

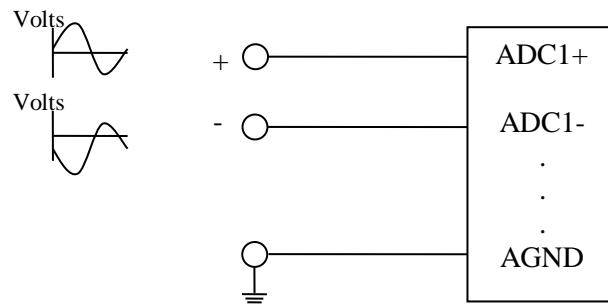


All dimensions are in inches. Drawings are not to scale.

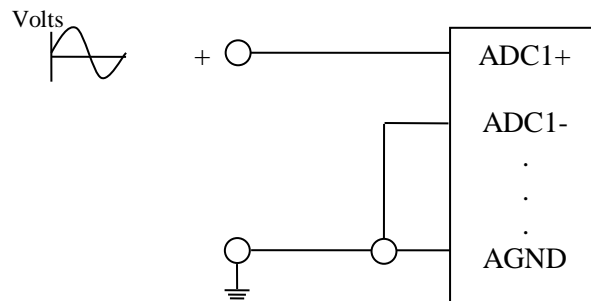


## Sample Wiring Diagram

### Differential Analog Input Signal



### Single-Ended Analog Input Signal



## P1: Backplane Bus

---

This connector is used for interface to UMAC's processor bus via the backplane of the 3U rack. The signals that are brought in through this connector are buffered on board.

## P3

---



*Caution*

Do not use this connector. This 20-pin header is for factory use only (calibration).

## TB1 (4-Pin Terminal Block)

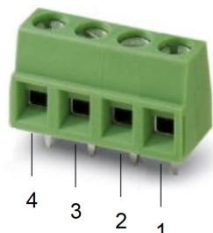
---



*Caution*

Do not use TB1 when the ACC-36E is plugged into the backplane.

This 4-pin terminal block provides the connection for an external power supply (standalone mode only).

<b>TB1: 4-Point Terminal Block</b>			
		4	3 2 1
Pin #	Symbol	Function	Description
1	GND	Common	Digital ground
2	+5V	Input	External supply
3	+15V	Input	External supply
4	-15V	Input	External supply

## DB15 Breakout Option

---

### J1/J2 Top, J1/J2 Bottom

Through these connectors, the analog signals are brought into ACC-36E. In addition, the  $\pm 12$  to 15 V power supplies are brought out. These power supplies may be used in situations where a separate supply unit is not available for the analog transducers.

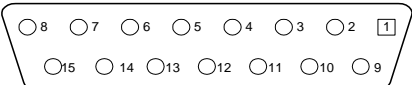


The two fuses limit the current drawn to 0.5 A on each supply line.

*Note*

---

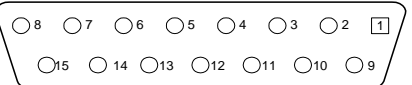
## J1 Top: ADC1 through ADC4

J1 Top: D-sub DA-15F Mating: D-sub DA-15M			
Pin #	Symbol	Function	Description
1	ADC1+	Input	+Analog Input #1
2	ADC2+	Input	+Analog Input #2
3	ADC3+	Input	+Analog Input #3
4	ADC4+	Input	+Analog Input #4
5	Open	N/A	
6	AGND	Common	Ground
7	+12V	Output	Positive supply
8	AGND-	Common	Ground
9	ADC1-	Input	-Analog Input #1
10	ADC2-	Input	-Analog Input #2
11	ADC3-	Input	-Analog Input #3
12	ADC4-	Input	-Analog Input #4
13	Open	N/A	
14	AGND	Common	Ground
15	-12V	Output	Negative supply

*Note*

- The  $\pm 12V$  output can be used for an external device (i.e. sensor). The drawn current should not exceed 0.5 Amperes.
- The common ground (pin #7) is tied to the digital ground of the UMAC rack.

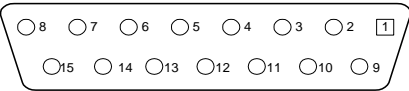
## J2 Top: ADC5 through ADC8

J2 Top: D-sub DA-15F Mating: D-sub DA-15M			
Pin #	Symbol	Function	Description
1	+ADC5	Input	+Analog Input #5
2	+ADC6	Input	+Analog Input #6
3	+ADC7	Input	+Analog Input #7
4	+ADC8	Input	+Analog Input #8
5	Open	N/A	
6	AGND	Common	Ground
7	+12V	Output	Positive supply
8	AGND	Common	Ground
9	-ADC5	Input	-Analog Input #5
10	-ADC6	Input	-Analog Input #6
11	-ADC7	Input	-Analog Input #7
12	-ADC8	Input	-Analog Input #8
13	Open	N/A	
14	AGND	Common	Ground
15	-12V	Output	Negative supply

*Note*

- The  $\pm 12V$  output can be used for an external device (i.e. sensor). The drawn current should not exceed 0.5 Amperes.
- The common ground (pin #7) is tied to the digital ground of the UMAC rack.

## J1 Bottom: ADC9 through ADC12

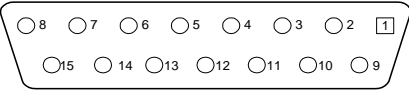
J1 Bottom: D-sub DA-15F Mating: D-sub DA-15M			
Pin #	Symbol	Function	Description
1	+ADC9	Input	+Analog Input #9
2	+ADC10	Input	+Analog Input #10
3	+ADC11	Input	+Analog Input #11
4	+ADC12	Input	+Analog Input #12
5	Open	N/A	
6	AGND	Common	Ground
7	+12V	Output	Positive supply
8	AGND	Common	Ground
9	-ADC9	Input	-Analog Input #9
10	-ADC10	Input	-Analog Input #10
11	-ADC11	Input	-Analog Input #11
12	-ADC12	Input	-Analog Input #12
13	Open	N/A	
14	AGND	Common	Ground
15	-12V	Output	Negative supply



Note

- The  $\pm 12V$  output can be used for an external device (i.e. sensor). The drawn current should not exceed 0.5 Amperes.
- The common ground (pin #7) is tied to the digital ground of the UMAC rack.

## J2 Bottom: ADC13 through ADC16

J2 Bottom: D-sub DA-15F Mating: D-sub DA-15M			
Pin #	Symbol	Function	Description
1	+ADC13	Input	+Analog Input #13
2	+ADC14	Input	+Analog Input #14
3	+ADC15	Input	+Analog Input #15
4	+ADC16	Input	+Analog Input #16
5	Open	N/A	
6	AGND	Common	Ground
7	+12V	Output	Positive supply
8	AGND	Common	Ground
9	-ADC13	Input	-Analog Input #13
10	-ADC14	Input	-Analog Input #14
11	-ADC15	Input	-Analog Input #15
12	-ADC16	Input	-Analog Input #16
13	Open	N/A	
14	AGND	Common	Ground
15	-12V	Output	Negative supply

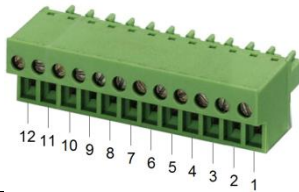


Note

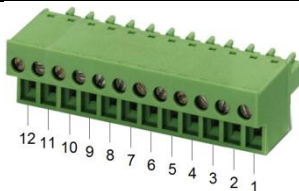
- The  $\pm 12V$  output can be used for an external device (i.e. sensor). The drawn current should not exceed 0.5 Amperes.
- The common ground (pin #7) is tied to the digital ground of the UMAC rack.

## Terminal Block Option

### TB1 Top: ADC1 through ADC4

<b>TB1 Top: 12-Point Terminal Block</b>			
Pin #	Symbol	Function	Description
1	ADC1+	Input	Analog Input #1
2	ADC1-	Input	Analog Input #1/
3	ADC2+	Input	Analog Input #2
4	ADC2-	Input	Analog Input #2/
5	ADC3+	Input	Analog Input #3
6	ADC3-	Input	Analog Input #3/
7	ADC4+	Input	Analog Input #4
8	ADC4-	Input	Analog Input #4/
9	NC	NC	
10	NC	NC	
11	AGND	Input/Output	Common reference for ADC1-ADC4
12	AGND	Input/Output	Common reference for ADC1-ADC4

### TB2 Top: ADC5 through ADC8

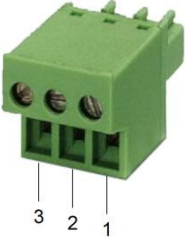
<b>TB2 Top: 12-Point Terminal Block</b>			
Pin #	Symbol	Function	Description
1	ADC5+	Input	Analog Input #5
2	ADC5-	Input	Analog Input #5/
3	ADC6+	Input	Analog Input #6
4	ADC6-	Input	Analog Input #6/
5	ADC7+	Input	Analog Input #7
6	ADC7-	Input	Analog Input #7/
7	ADC8+	Input	Analog Input #8
8	ADC8-	Input	Analog Input #8/
9	NC	NC	
10	NC	NC	
11	AGND	Input/Output	Common reference for ADC5-ADC8
12	AGND	Input/Output	Common reference for ADC5-ADC8



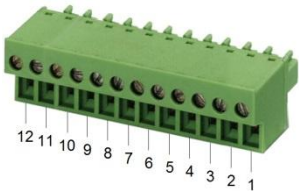
#### Note

Tie the ADC- pin to ground if using single-ended wiring to ensure full resolution and proper operation of the ADC channel.

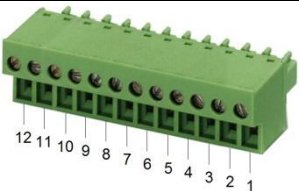
**TB3 Top: Power Supply Outputs**

<b>TB3 Top: 3-Point Terminal Block</b>				
Pin #	Symbol	Function	Description	Notes
1	+15V	Output	+15 V from UMAC power supply	Fused (1/2 A)
2	-15V	Output	-15 V from UMAC power supply	Fused (1/2 A)
3	AGND	Input/Output	Common reference for ADC1-ADC16	

**TB1 Bottom: ADC9 through ADC12**

<b>TB1 Bottom: 12-Point Terminal Block</b>			
Pin #	Symbol	Function	Description
1	ADC9+	Input	Analog Input #9
2	ADC9-	Input	Analog Input #9/
3	ADC10+	Input	Analog Input #10
4	ADC10-	Input	Analog Input #10/
5	ADC11+	Input	Analog Input #11
6	ADC11-	Input	Analog Input #11/
7	ADC12+	Input	Analog Input #12
8	ADC12-	Input	Analog Input #12/
9	NC	NC	
10	NC	NC	
11	AGND	Input/Output	Common reference for ADC9-ADC12
12	AGND	Input/Output	Common reference for ADC9-ADC12

**TB2 Bottom: ADC13 through ADC16**

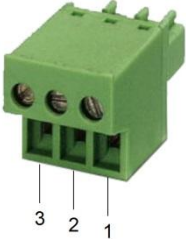
<b>TB2 Bottom: 12-Point Terminal Block</b>			
Pin #	Symbol	Function	Description
1	ADC13+	Input	Analog Input #13
2	ADC13-	Input	Analog Input #13/
3	ADC14+	Input	Analog Input #14
4	ADC14-	Input	Analog Input #14/
5	ADC15+	Input	Analog Input #15
6	ADC15-	Input	Analog Input #15/
7	ADC16+	Input	Analog Input #16
8	ADC16-	Input	Analog Input #16/
9	NC	NC	
10	NC	NC	
11	AGND	Input/Output	Common reference for ADC13-ADC16
12	AGND	Input/Output	Common reference for ADC13-ADC16

**Note**

Tie the ADC- pin to ground if using single-ended wiring to ensure full resolution and proper operation of the ADC channel.



**TB3 Bottom: Power Supply Outputs**

<b>TB3 Bottom: 3-Point Terminal Block</b>			
		3	2 1
Pin #	Symbol	Function	Description
1	NC	-	Do not connect
2	NC	-	Do not connect
3	NC	-	Do not connect

## JCAL 20-Pin Header Connector

Pin #	Symbol	Function	Description	Notes
1	ANAI00	Input	Analog Input 1	0-10 V or +/-10 V range
2	ANAI01	Input	Analog Input 2	0-10 V or +/-10 V range
3	ANAI02	Input	Analog Input 3	0-10 V or +/-10 V range
4	ANAI03	Input	Analog Input 4	0-10 V or +/-10 V range
5	ANAI04	Input	Analog Input 5	0-10 V or +/-10 V range
6	ANAI05	Input	Analog Input 6	0-10 V or +/-10 V range
7	ANAI06	Input	Analog Input 7	0-10 V or +/-10 V range
8	ANAI07	Input	Analog Input 8	0-10 V or +/-10 V range
9	ANAI08	Input	Analog Input 9	0-10 V or +/-10 V range
10	ANAI09	Input	Analog Input 10	0-10 V or +/-10 V range
11	ANAI10	Input	Analog Input 11	0-10 V or +/-10 V range
12	ANAI11	Input	Analog Input 12	0-10 V or +/-10 V range
13	ANAI12	Input	Analog Input 13	0-10 V or +/-10 V range
14	ANAI13	Input	Analog Input 14	0-10 V or +/-10 V range
15	ANAI14	Input	Analog Input 15	0-10 V or +/-10 V range
16	ANAI15	Input	Analog Input 16	0-10 V or +/-10 V range
17	REF1-	Input		Reference to AGND
18	REF2-	Input		Reference to AGND
19	AGND	Common		
20	AGND	Common		



### Note

This header is for internal use only, and is not pin-for-pin compatible with the PMAC2 JANA port.

## DECLARATION OF CONFORMITY

---

**Application of Council Directive: 89/336/EEC, 72/23/EEC**

**Manufacturers Name:** Delta Tau Data Systems, Inc.

**Manufacturers Address:** 21314 Lassen Street  
Chatsworth, CA 91311  
USA

We, Delta Tau Data Systems, Inc., hereby declare that the product

**Product Name:** Accessory 36E

**Model Number:** 603483

And all of its options conforms to the following standards:

EN61326: 1997	Electrical equipment for measurement, control, and laboratory use- EMC requirements
EN55011: 1998	Limits and methods of measurements of radio disturbance characteristics of information technology equipment
EN61010-1	Electrical equipment for measurement, control, and laboratory use- Safety requirements
EN61000-3-2 :1995 A14:1998	Limits for harmonic current emissions. Criteria A
EN61000-3-3: 1995	Limitation of voltage fluctuations and flicker in low-voltage supply systems for equipment with rated current $\leq 16A$ . Criteria B.
EN61000-4-2:1995 A1: 1998	Electro Static Discharge immunity test. Criteria B
EN61000-4-3: 1995 A1: 1998	Radiated, radio-frequency, electromagnetic field immunity test. Criteria A
EN61000-4-4: 1995	Electrical fast transients/burst immunity test. Criteria B
EN61000-4-5: 1995	Surge Test. Criteria B
EN61000-4-6: 1996	Conducted immunity test. Criteria A
EN61000-4-11: 1994	Voltage dips test. Criteria B and C

**Date Issued:** 11 May 2006

**Place Issued:** Chatsworth, California USA

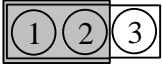
*Yolande Cano*

Yolande Cano  
Quality Assurance Manager

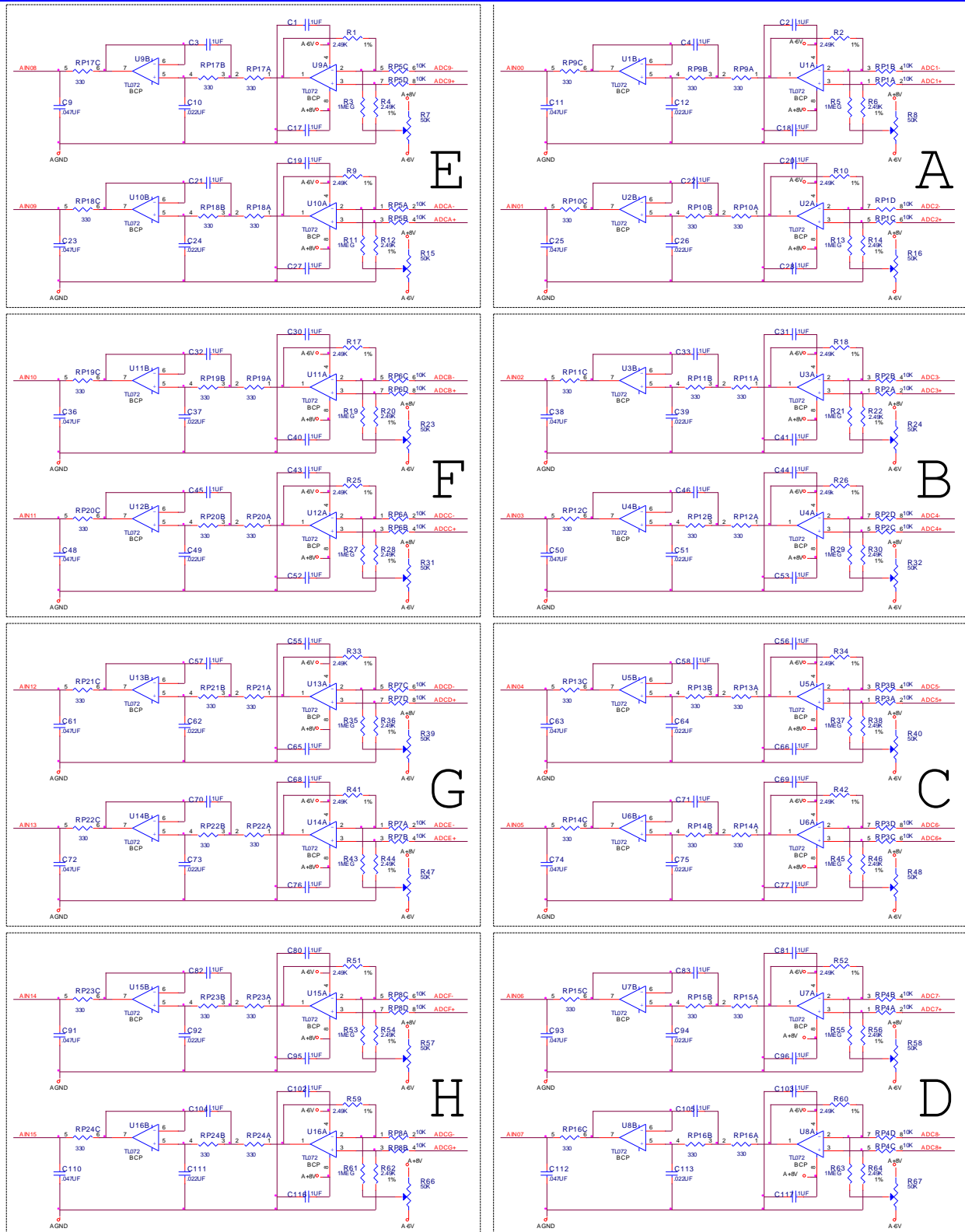
Mark of Compliance



## APPENDIX A: E-POINT JUMPERS

Jumper	Configuration	Default
<b>E1</b> 	Jump pins 1 to 2 for all Turbo UMAC CPUs, for all MACRO 16 CPUs, and for MACRO8 CPUs Rev. 104 and newer	Set by factory
	Jump pins 2 to 3 for MACRO8 CPUs Rev. 103 and older	

## APPENDIX B: SCHEMATICS



## APPENDIX C: USING POINTERS

Below are alternate methods for accessing the data structures of Power PMAC in order to read ADCs on the ACC-36E.

### Manual ADC Read Using Pointers

Following are the necessary steps for implementing the manual ADC read method. The example parameters given here are for an ACC-36E at I/O base address offset \$A00000:

1. Point an available M-Variable (12-bit wide) to bits 8-19 of the ACC-36E I/O base address offset, and another to bits 20-31. These are the “**Data Read**” registers of the selected pair. Bits 8-19 hold the result for channels 1-8; bits 20-31 hold the results of channels 9-16. They can be defined as follows:

Unsigned (positive input voltage only) for Unipolar Mode:

```
ptr UnipolarDataReadLow->u.io:$A00000.8.12;           // Read Data, channels 1-8, unipolar
ptr UnipolarDataReadHigh->u.io:$A00000.20.12;          // Read Data, channels 9-16, unipolar
```

Signed (negative/positive input voltage) for Bipolar Mode:

```
ptr BipolarDataReadLow->s.io:$A00000.8.12;           // Read Data, channels 1-8, bipolar
ptr BipolarDataReadHigh->s.io:$A00000.20.12;          // Read Data, channels 9-16, bipolar
```

2. Point an available M-Variable (24-bit wide unsigned) to the base address of the ACC-36E. This is the “**Channel Select**” Pointer.

```
ptr ChannelSelect->u.io:$A00000.8.24                  // Channel Select
```

3. Point an available M-Variable (1-bit wide unsigned) to the Low ADC Ready bit of the ACC-36E, and another 1-bit wide unsigned M-Variable to the high ADC Ready bit. These are the “**ADC Ready**” Pointers.

```
ptr LowADCReady->u.io:$D00180.13.1;                  // Low ADC Ready Bit (for channels 1-8)
ptr HighADCReady->u.io:$D00184.13.1;                  // High ADC Ready Bit (for channels 9-16)
```

The Low and High ADC Ready bits are at bit 13 of the ADC Ready Offsets shown in the following table:

Index (n)	Base Offset	Low ADC Ready Offset	High ADC Ready Offset
0	\$A00000	\$D00180	\$D00184
4	\$A08000	\$D08180	\$D08184
8	\$A10000	\$D10180	\$D10184
12	\$A18000	\$D18180	\$D18184
1	\$B00000	\$D00190	\$D00194
5	\$B08000	\$D08190	\$D08194
9	\$B10000	\$D10190	\$D10194
13	\$B18000	\$D18190	\$D18194
2	\$C00000	\$D001C0	\$D001C4
6	\$C08000	\$D081C0	\$D081C4
10	\$C10000	\$D101C0	\$D101C4
14	\$C18000	\$D181C0	\$D181C4
3	\$D00000	\$D001D0	\$D001D4
7	\$D08000	\$D081D0	\$D081D4
11	\$D10000	\$D101D0	\$D101D4
15	\$D18000	\$D181D0	\$D181D4

4. Using the **Channel Select** Pointer, specify the pair and voltage mode desired, as follows:

		Value to Which to Set Channel Select Pointer	
Pair #	ADC Channels	Unipolar Inputs	Bipolar Inputs
1	1 & 9	0	8
2	2 & 10	1	9
3	3 & 11	2	10
4	4 & 12	3	11
5	5 & 13	4	12
6	6 & 14	5	13
7	7 & 15	6	14
8	8 & 16	7	15

5. Wait until the **ADC Ready** bits become 1, then read and/or copy the data contained in the **Data Read** registers described above.

### ADC Manual Read Example PLCs

Ultimately, the above procedure can be implemented in a PLC script to read all channels consecutively and consistently, creating a “custom automatic” function. See the following examples.

## Unipolar Script PLC Example

Configuring a single ACC-36E set at base offset \$A00000, selects and reads pairs 1 through 8 successively as unipolar in a PLC, and then stores the results.

```
ptr UnipolarDataReadLow->u.io:$A00000.8.12; // Data Read, channels 1-8, unsigned for unipolar
ptr UnipolarDataReadHigh->u.io:$A00000.20.12; // Data Read, channels 9-16, unsigned for unipolar
ptr ChSelect->u.io:$A00000.8.24; // Channel Select Pointer
ptr LowADCReady->u.io:$D00180.13.1; // Low ADC Ready Bit (for channels 1-8)
ptr HighADCReady->u.io:$D00184.13.1; // High ADC Ready Bit (for channels 9-16)

global ADC1; // Channel 1 ADC storage
global ADC2; // Channel 2 ADC storage
global ADC3; // Channel 3 ADC storage
global ADC4; // Channel 4 ADC storage
global ADC5; // Channel 5 ADC storage
global ADC6; // Channel 6 ADC storage
global ADC7; // Channel 7 ADC storage
global ADC8; // Channel 8 ADC storage
global ADC9; // Channel 9 ADC storage
global ADC10; // Channel 10 ADC storage
global ADC11; // Channel 11 ADC storage
global ADC12; // Channel 12 ADC storage
global ADC13; // Channel 13 ADC storage
global ADC14; // Channel 14 ADC storage
global ADC15; // Channel 15 ADC storage
global ADC16; // Channel 16 ADC storage

Open PLC 1
ChSelect = 0; // Select Channels 1 & 9, unipolar
While(LowADCReady != 1 && HighADCReady != 1){} // Wait for conversions to finish
ADC1 = UnipolarDataReadLow; // Read and copy result into storage
ADC9 = UnipolarDataReadHigh; // Read and copy result into storage

ChSelect = 1; // Select Channels 2 & 10, unipolar
While(LowADCReady != 1 && HighADCReady != 1){} // Wait for conversions to finish
ADC2 = UnipolarDataReadLow; // Read and copy result into storage
ADC10 = UnipolarDataReadHigh; // Read and copy result into storage

ChSelect = 2; // Select Channels 3 & 11, unipolar
While(LowADCReady != 1 && HighADCReady != 1){} // Wait for conversions to finish
ADC3 = UnipolarDataReadLow; // Read and copy result into storage
ADC11 = UnipolarDataReadHigh; // Read and copy result into storage

ChSelect = 3; // Select Channels 4 & 12, unipolar
While(LowADCReady != 1 && HighADCReady != 1){} // Wait for conversions to finish
ADC4 = UnipolarDataReadLow; // Read and copy result into storage
ADC12 = UnipolarDataReadHigh; // Read and copy result into storage

ChSelect = 4; // Select Channels 5 & 13, unipolar
While(LowADCReady != 1 && HighADCReady != 1){} // Wait for conversions to finish
ADC5 = UnipolarDataReadLow; // Read and copy result into storage
ADC13 = UnipolarDataReadHigh; // Read and copy result into storage

ChSelect = 5; // Select Channels 6 & 14, unipolar
While(LowADCReady != 1 && HighADCReady != 1){} // Wait for conversions to finish
ADC6 = UnipolarDataReadLow; // Read and copy result into storage
ADC14 = UnipolarDataReadHigh; // Read and copy result into storage

ChSelect = 6; // Select Channels 7 & 15, unipolar
While(LowADCReady != 1 && HighADCReady != 1){} // Wait for conversions to finish
ADC7 = UnipolarDataReadLow; // Read and copy result into storage
ADC15 = UnipolarDataReadHigh; // Read and copy result into storage

ChSelect = 7; // Select Channels 8 & 16, unipolar
While(LowADCReady != 1 && HighADCReady != 1){} // Wait for conversions to finish
ADC8 = UnipolarDataReadLow; // Read and copy result into storage
ADC16 = UnipolarDataReadHigh; // Read and copy result into storage
Close
```



## Bipolar Script PLC Example

This example selects and reads pairs 1 through 8 successively as bipolar in a PLC and stores the results.

```
ptr BipolarDataReadLow->s.io:$A00000.8.12;    // Data Read, channels 1-8, signed for bipolar
ptr BipolarDataReadHigh->s.io:$A00000.20.12;   // Data Read, channels 9-16, signed for bipolar
ptr ChSelect->u.io:$A00000.8.24;              // Channel Select Pointer
ptr LowADCReady->u.io:$D00180.13.1;           // Low ADC Ready Bit (for channels 1-8)
ptr HighADCReady->u.io:$D00184.13.1;          // High ADC Ready Bit (for channels 9-16)

global ADC1;    // Channel 1  ADC storage
global ADC2;    // Channel 2  ADC storage
global ADC3;    // Channel 3  ADC storage
global ADC4;    // Channel 4  ADC storage
global ADC5;    // Channel 5  ADC storage
global ADC6;    // Channel 6  ADC storage
global ADC7;    // Channel 7  ADC storage
global ADC8;    // Channel 8  ADC storage
global ADC9;    // Channel 9  ADC storage
global ADC10;   // Channel 10 ADC storage
global ADC11;   // Channel 11 ADC storage
global ADC12;   // Channel 12 ADC storage
global ADC13;   // Channel 13 ADC storage
global ADC14;   // Channel 14 ADC storage
global ADC15;   // Channel 15 ADC storage
global ADC16;   // Channel 16 ADC storage

Open PLC 1
ChSelect = 8;                                // Select ADC Channels 1 & 9, bipolar
While(LowADCReady != 1 && HighADCReady != 1){} // Wait for conversions to finish
ADC1 = BipolarDataReadLow;                    // Read and copy result into storage
ADC9 = BipolarDataReadHigh;                   // Read and copy result into storage

ChSelect = 9;                                // Select ADC Channels 2 & 10, bipolar
While(LowADCReady != 1 && HighADCReady != 1){} // Wait for conversions to finish
ADC2 = BipolarDataReadLow;                    // Read and copy result into storage
ADC10 = BipolarDataReadHigh;                  // Read and copy result into storage

ChSelect = 10;                               // Select ADC Channels 3 & 11, bipolar
While(LowADCReady != 1 && HighADCReady != 1){} // Wait for conversions to finish
ADC3 = BipolarDataReadLow;                    // Read and copy result into storage
ADC11 = BipolarDataReadHigh;                  // Read and copy result into storage

ChSelect = 11;                               // Select ADC Channels 4 & 12, bipolar
While(LowADCReady != 1 && HighADCReady != 1){} // Wait for conversions to finish
ADC4 = BipolarDataReadLow;                    // Read and copy result into storage
ADC12 = BipolarDataReadHigh;                  // Read and copy result into storage

ChSelect = 12;                               // Select ADC Channels 5 & 13, bipolar
While(LowADCReady != 1 && HighADCReady != 1){} // Wait for conversions to finish
ADC5 = BipolarDataReadLow;                    // Read and copy result into storage
ADC13 = BipolarDataReadHigh;                  // Read and copy result into storage

ChSelect = 13;                               // Select ADC Channels 6 & 14, bipolar
While(LowADCReady != 1 && HighADCReady != 1){} // Wait for conversions to finish
ADC6 = BipolarDataReadLow;                    // Read and copy result into storage
ADC14 = BipolarDataReadHigh;                  // Read and copy result into storage

ChSelect = 14;                               // Select ADC Channels 7 & 15, bipolar
While(LowADCReady != 1 && HighADCReady != 1){} // Wait for conversions to finish
ADC7 = BipolarDataReadLow;                    // Read and copy result into storage
ADC15 = BipolarDataReadHigh;                  // Read and copy result into storage

ChSelect = 15;                               // Select ADC Channels 8 & 16, bipolar
While(LowADCReady != 1 && HighADCReady != 1){} // Wait for conversions to finish
ADC8 = BipolarDataReadLow;                    // Read and copy result into storage
ADC16 = BipolarDataReadHigh;                  // Read and copy result into storage
Close
```

## Unipolar and Bipolar CPLC Example

Configuring one ACC-36E set at base offset \$A00000 and one set at \$B00000. It incorporates two functions, **ACC36E\_ADC** and **ACC36E\_WaitForADC**, which take user parameters about the card index, pair selection, and signal polarity, and then return ADC results through pointers. The benefit of using these functions is that the user does not need to program in the actual card base address, but only the corresponding index (*n*); PMAC will then determine the appropriate addressing automatically using the **Sys.OffsetCardIO[n]** structure for base offset and **Sys.OffsetCardIOCid[n]** for the ready bit offsets. Please read the comments under the function definitions for more details on how to use the functions.



### WARNING

Make sure to use the **ACC36E\_WaitForADC** function, which has additional precautions in order to release control of the loop in the event that the conversion bits never become 1, to prevent lockup and loss of control of the machine PMAC is controlling.

```
#include <RtGpShm.h>
#include <stdio.h>
#include <dlfcn.h>

// Definition(s)
#define Card1Index    0           // For base offset $A00000
#define Card2Index    1           // For base offset $B00000
#define Unipolar_Code 0           // For unipolar input signals
#define Bipolar_Code  1           // For bipolar input signals

// Prototype(s)
int ACC36E_ADC(unsigned int Card_Index, unsigned int ADC_Pair, unsigned int Polarity, int
*ADC_Low, int *ADC_High);
int ACC36E_WaitForADC(unsigned int Card_Index);

void user_plcc() {
    unsigned int ADCuResult[16],index,ADCuLow,ADCuHigh,Channel_Number;
    int ADCsResult[16],ADCsLow,ADCsHigh,ErrorCode;
    for(index = 0; index < 8; index++) {
        Channel_Number = index + 1; // Compute channel number

        // Request ADC results, unsigned
        ErrorCode =
ACC36E_ADC(Card1Index,Channel_Number,Unipolar_Code,(int*)&ADCuLow,(int*)&ADCuHigh);
        if(ErrorCode < 0) {
            return; // error
        }
        ADCuResult[index] = ADCuLow;           // Store low result in array
        ADCuResult[index + 8] = ADCuHigh;      // Store high result in array

        // Request ADC results, signed
        ErrorCode =
ACC36E_ADC(Card2Index,Channel_Number,Bipolar_Code,&ADCsLow,&ADCsHigh);
        if(ErrorCode < 0)
        {
            return; // error
        }
        ADCsResult[index] = ADCsLow;           // Store low result in array
        ADCsResult[index + 8] = ADCsHigh;      // Store high result in array

        // Store results in P-Variables so they are accessible in motion programs
        // (this step is optional)
        pshm->P[5000 + index] = ADCuResult[index];
        pshm->P[5000 + index + 8] = ADCuResult[index + 8];
        pshm->P[6000 + index] = ADCsResult[index];
        pshm->P[6000 + index + 8] = ADCsResult[index + 8];
    }
    return;
}
```

```

int ACC36E_ADC(unsigned int Card_Index, unsigned int ADC_Pair, unsigned int Polarity, int
*ADC_Low, int *ADC_High)
{
    /*Inputs:
    Card_Index: The addressing index (n) of the card, based on SW1 settings.
    ADC_Pair: The number of the ADC pair one desires to sample. Pair 1: ADC 1 & 9, pair 2:
    ADC 2 & 10
    ... pair 8: ADC 8 & 16
    Polarity: 0 = sample as unipolar input signal, 1 = sample as bipolar input signal
    *ADC_Low: Address of the variable the user wants to use to store the low ADC result (user
    needs
    to pass in &ADC_Low)
    *ADC_High: Address of the variable the user wants to use to store the high ADC result
    (user needs to pass in &ADC_High)

    Outputs:
    return 0 if ADC conversion was successful
    return -1 if user entered invalid Card_Index
    return -2 if user entered invalid ADC_Pair
    return -3 if user entered invalid Polarity
    return -4 if ADC conversion timed out
    Function puts ADC result low into *ADC_Low and result high into *ADC_High if the
    conversion was successful*/
    unsigned int Address,BaseOffset,ConvertCode;
    int WaitResult;
    volatile unsigned int *pACC36E_ADC_ChSelect;
    volatile unsigned int *pACC36E_Data_Read_Unipolar; // Unsigned for bipolar input signals
    volatile int *pACC36E_Data_Read_Bipolar; // Signed for bipolar input signals
    if(Card_Index < 0 || Card_Index > 15)
    {
        return -1;
    }
    if((ADC_Pair < 1) || (ADC_Pair > 8))
    {
        return -2;
    }
    if((Polarity != 0) && (Polarity != 1))
    {
        return -3;
    }
    if(Polarity == 0) // Unipolar input signal
    {
        ConvertCode = ADC_Pair - 1;
    } else { // Bipolar input signal
        ConvertCode = ADC_Pair + 7;
    }
    BaseOffset = pshm->OffsetCardIO[Card_Index];
    Address = (unsigned int)piom + BaseOffset/4;
    pACC36E_ADC_ChSelect = (volatile unsigned int*)Address;
    // Shift and mask to write the convert code to the correct place in the ADC word
    *pACC36E_ADC_ChSelect = ((ConvertCode) << 8) & 0xFFFFF00;
    // Wait for the ADC to finish converting
    WaitResult = ACC36E_WaitForADC(Card_Index);
    if(WaitResult == -1){ // If the ADC conversion timed out
        return (-4); // Return with error code
    } else { // Otherwise return ADC result
        if(Polarity == 0)
        {
            pACC36E_Data_Read_Unipolar = (volatile unsigned int*)Address;
            // Shift and cast to get just the ADC results with the proper signs
            *ADC_Low = ((unsigned int)((*pACC36E_Data_Read_Unipolar) << 12) >> 20));
            *ADC_High = ((unsigned int)((*pACC36E_Data_Read_Unipolar) >> 20));
        } else {
            pACC36E_Data_Read_Bipolar = (volatile int*)Address;
            // Shift and cast to get just the ADC results with the proper signs
            *ADC_Low = ((int)((*pACC36E_Data_Read_Bipolar) << 12) >> 20));
            *ADC_High = ((int)((*pACC36E_Data_Read_Bipolar) >> 20));
        }
    }
    return 0;
}

```

```

int ACC36E_WaitForADC(unsigned int Card_Index)
{
    // Waits until ADC conversions have completed
    // Inputs:
    // Card_Index: index (n) from POWER section of Addressing ACC-36E table

    // Outputs:
    // returns 0 if successfully performed ADC conversion
    // returns -1 if conversion did not complete within Timeout ms
    volatile unsigned int *pRdyLow,*pRdyHigh;
    unsigned int RdyLow = 0,RdyHigh = 0,iterations = 0;
    double Present_Time,Conversion_Start_Time,Time_Difference,Timeout,Timeout_us;
    struct timespec SleepTime={0};
    SleepTime.tv_nsec=1000000;
    // Get time at (almost) start of conversion (microseconds)
    Conversion_Start_Time = GetCPUClock();
    // Timeout: Maximum permitted time to wait for ADC conversion to finish
    // before error (milliseconds)
    Timeout = 500; // Milliseconds
    Timeout_us = Timeout*1000; // Convert to microseconds
    // Point to word containing low ADC ready bit
    pRdyLow = piom + (pshm->OffsetCardIOCid[Card_Index]/4);
    // Point to word containing high ADC ready bit
    pRdyHigh = piom + ((pshm->OffsetCardIOCid[Card_Index] + 4)/4);
    do
    {
        // If the loop has taken a multiple of 50 iterations to finish
        if(iterations == 50)
        {
            // Release control for 1 ms so PMAC does not go into Watchdog mode while
            // waiting for conversion to finish
            nanosleep(&SleepTime,NULL); // Release thread and wait 1 msec

            iterations = 0; // Reset iteration counter
        }
        Present_Time = GetCPUClock(); // Obtain current system time
        // Compute difference in time between starting conversion and now
        Time_Difference = Present_Time-Conversion_Start_Time;
        if(Time_Difference > Timeout_us) // If more than Timeout ms have elapsed
        {
            return (-1); // Return with error code
        }
        // Shift and cast to get just 13th bit (the ADC low ready bit)
        RdyLow = (unsigned int)((*pRdyLow << 18) >> 31);
        // Shift and cast to get just 13th bit (the ADC high ready bit)
        RdyHigh = (unsigned int)((*pRdyHigh << 18) >> 31);
        iterations++;
    } while(RdyLow != 1 && RdyHigh != 1); // Test ADC ready bit
    return 0; // Return with success code
}

```